

# **A Tool to Randomly Allocate Projects to Students and Advisors**

**John Harrow**

**Supervised by Dr. Craig Ramsay**

**MSc Advanced Computer Science Project**

**20/8/2025**

## **Acknowledgements**

I would like to thank my project advisor, Dr. Craig Ramsay, for their guidance, feedback, and support throughout the course of this project. Their advice was invaluable in helping me shape the direction of the work and in improving the quality of both the application and this report.

## **Abstract**

Effective allocation of students to their final year projects is a regular challenge found within academic institutions. Having to do the allocations manually is time-consuming, prone to errors and inconsistencies, and can become increasingly difficult as class sizes and range of projects increase. This project attempts to solve this issue by designing and developing a web app that automates and optimises the process of matching students to the projects they want.

The app was developed using Python and the Streamlit framework which allows Python code to be used to create web apps. It has a user friendly interface which allows for the uploading of data and for the matches to be generated. Three algorithms were chosen to create matches which each use slightly different strategies; Greedy Matching, Stable Marriage, and Linear Programming. Each algorithm takes the constraints outlined by the data set and configuration within the app to get the best possible result based on student preference.

The app does validation of the data when it is uploaded and lets the user know when there are any issues such as missing entries before moving on to doing the matching process. The results of the allocation are then summarised using tables and visual charts to show things like satisfactions scores and choice distributions. The results can then be downloaded as an excel workbook for further changes to be made or to be sent out to students.

Evaluation of the app was done using both made up data sets and more realistic data sets based of previous years allocation processes. It was found that Linear Programming usually results in the highest satisfaction score but having the app generate and export all three results to one excel workbook allows for the administrator to pick with result they want to work with.

The completed app offers a tool to administrators that is configurable and allows for analysis which helps them manage the allocation process more effectively. The app could be used as a foundation for more improvements to be done such as increased flexibility of data sets and more options for configuration.

## **Introduction**

Allocating students to projects during their final year is an important but time consuming and error prone admin task for overseers of courses. This allocation process can have a big impact on the experience of students as it can effect their engagement and enjoyment of completing the project. It can also impact the workload of staff members who are advising the projects as it is important for them not to be overloaded so they can give students the support they need. Students are usually given a list of available projects to choose from and are to give their preferred projects in rank order. The list of projects is created by collating all of the project ideas all of the participating members of staff have and making the students aware that there is limited availability for each project. The allocation process must balance between student preference, the supervisors availability, the project capacities and fairness between students which is a complex task.

Initially this process may seem quite straightforward but it becomes more difficult when constraints are added such as ensuring each supervisor does not go over their specified capacity and that none of the top three student choices can be from the same supervisor. Fairness also can become an issue as it may concern some students if they are getting one of their lower choices compared to another student getting their top choice. These factors make the process more challenging as not only does it require complex algorithms to do the matching but it has to also be perceived as fair.

In the past it has been normal for academic administrators to use manual methods such as using a spreadsheet with simple rules to manage the process. While this process can be great for smaller sized data sets and simpler constraints, it can become time consuming and error prone when the number of students and projects increase along with the constraints. Using a manual process does not allow the same ability to scale as an automated and can result in students not getting great matches when it was possible for them to do so. manual systems also do not have tools for evaluating the quality of the final matches which means it is more difficult for them to make sure the process has been done effectively and fairly.

Noticing these limits of the manual process, this project has set out to create a web app that assists with the allocation process by automating the initial matches. The goal was to design a tool that would accept data such as the student choices and project information, and return an optimised set of matches that can be analysed and exported. The app would not only automate the process but provide insight into how the results were created which allows for the user to evaluate and adjust any matches before they are finalised.

The system uses three different algorithms to try and vary the outcomes while still being close to optimum:

- Greedy Matching: This algorithm processes the students in a random order and attempts to assign each student to their top available choice. It is a simple and fast algorithm but it does not guarantee an optimum or fair outcome.
- Stable Marriage: This is adapted from the Gale-Shapely algorithm (Gale and Shapley, 1962) which ensures that no pair of student and advisor would prefer a different pairing than the currently assigned one. This process results in more fairness but can lead to unallocated students if the capacity is highly constrained.
- Linear Programming (LP): This algorithm treats the matching process as a mathematical optimisation task which has the goal of maximising the total satisfaction score across the data set of student choices. LP looks at all student choices at the same time so it requires more computation than the other two but usually is not an issue when the data sets are a normal size.

Using all three algorithms at the same time to create an output of matches allows for the comparison of outcomes such as the number of students that got their first choice, the average satisfaction score and the supervisor workloads. This lets the user decide which allocation they want to go with based on what they can determine from the analysis.

The app was developed using Python while using the Streamlit framework which was chosen because it makes it quite simple to turn Python code into a web app with an interface. Libraries such as PuLP (which is used for linear programming) and pandas (used for data manipulation). Matplotlib was also used for creating charts, and OpenPyXL was used for excel exporting. Using these allowed for a web based app to be developed that doesn't require being installed or any knowledge of programming to be used effectively which makes it suitable for academic staff in different areas of expertise.

The app contains the following core functional features:

- Uploading files in the form of either an excel workbook or multiple different CSV files containing the student choices, project info, advisor info and pre-allocated students.
- Running all of the three algorithms to create the allocations and show the results in a table together.
- Showing useful information such as student satisfaction score, project popularity and supervisors load distribution.
- It clearly shows which students have not been assigned within their top 3 so that alterations can be made if necessary.
- Makes user aware of any capacity issues before doing the allocation.
- Exporting the final allocations to an excel workbook where further editing or admin can be done.

To make sure that the application did not contain any errors or issues when processing the data some evaluation was done using some made up data sets and some based on the real world data from previous years. These tests resulted in some valuable findings such as that Linear Programming usually delivered the higher student satisfaction score. However it also revealed that no algorithm performs optimally in all circumstances.

The app also includes data visualisation so that the user can understand better what the data is telling them rather than just displaying the raw allocation data output. This allows for the user to understand and compare the performance of each algorithm so that they can make a more informed choice.

When developing the app, the focus was not only on how well the algorithms perform but on the user experience. The app was designed to have an interface that includes expandable analysis sections, clear warnings when there are issues with the input data and for the allocation data to be fully collated in a form that can be downloaded and useful to the user. Including error handling and data validation to make sure that any issues with the input data are noticed by the user before the allocations are made ensure that there aren't any mistaken allocations made which would have a negative impact on the whole allocation process.

The main objectives of this project that were decided at the start were to:

- Automate the allocation of students to their preferred choices while following the constraints set out.
- Use multiple algorithms to allow for comparison of allocation results.
- To allow for analysis of the results by using charts and tables to make clear the allocation outcomes.
- For the app to be a web based so it can easily be accessed by users.

- To be able to export the final allocations so that they can be further altered or sent out to students.

These objectives have been achieved and the resulting prototype has been deployed, tested and has the foundation for further development to be made.

This report documents the process of the development of the app from identifying the issues with the previous methods of allocation, researching strategies to improve the process using algorithms, through to designing, implementing and evaluating the app. By the end of the report, the reader should have gained an understanding of how the system works, why decisions about the app were made and what trade-offs were considered. The app is tailored for project allocation in the university setting, however the approaches used are relevant to many other fields that require assignment or scheduling based on choice preference and constraints.

## Background Research

Allocating students to their final year projects is a regular admin challenge for a lot of university staff. The task usually involves students giving their preferences for which projects they would like and the administrator of the allocation process must balance these preferences against the many constraints that come into play. These constraints can be things like supervisors capacity, number of students per project, and any other specific limitation that is required for the process.

In the past this kind of problem has been solved manually using things like spreadsheets which works well if the data set is smaller but it can run into issues when a larger number of students or projects is required to be matched. Having the process mostly automated can reduce the chance of error and make the process more efficient and less time consuming for the administrator.

Insight into how the allocation process is currently done manually was gained by meeting with the project allocation coordinator and getting them to go step by step through their current process. It begins with the coordinator emailing all of the advisors to get project ideas from them. Once these have all been received and collated into a spreadsheet that links to more information about each project which students will have access to so that they can learn about each project they might want to do.

A form is then created using MyDundee which allows students to submit their top 3 choices or propose a project that they have discussed with an advisor who has agreed that the project is suitable. The coordinator then has to individually go through each of the responses from students and add their information and choices to a spreadsheet while double checking that the information entered by the student is correct and doesn't contain any errors such as having two projects from the same advisor.

It is then time for the coordinator to match the student choices to available projects but before that can start any prearranged projects or ones that involve internships must be matched first. After those have been matched each student is manually gone through and matched with their first choice if there is availability, if not they are allocated their second or third. This will then leave the coordinator with a spreadsheet which contains the allocations of students to projects which ideally are as close to the students first choices as possible.

A few main issues and pain points were identified in the current process. One being that having to manually transfer the student responses into a data structure is time consuming and error prone. It is easy for the coordinator to make a mistake when copying or to not notice an error with what has been submitted. Another being having to manually validate the responses to make sure that multiple projects from the same advisor have not been chosen and that the responses are actually valid choices. Another

being manually allocating students to projects as it is difficult to allocate fairly and accurately when there are a large number of students or when deciding which students get their first choice when projects are at full capacity.

These issues cause the coordinator to use up a lot of their time and can allow for unnecessary mistakes to be made during the assignment of projects which will negatively affect student experience by not allowing them to get the best choice they could have.

From the issues that were identified ideas for how to solve them were generated. To reduce the time consuming and error prone nature of manually transferring the choices from students into a data structure, the input parsing and validation could be automated. This would allow for the coordinator to send out the choice forms to students, receive the responses and have them already validated and organised in a data structure ready to be matched to projects without any manual input. To improve the quality and fairness of matching students to projects a matching algorithm can be used. This will not only save time by having the initial matching done quickly without any effort from the coordinator, it can also reduce errors and make sure it has picked optimal choices to maximise the chance students get their best choice.

Common algorithmic approaches to this kind of problem are:

#### **Stable Matching Algorithms:**

The Stable Marriage Problem (SMP) was introduced by Gale and Shapley in 1962 as a way to match two groups of the same size based on preference lists. A matching is considered stable if there are no two participants who would prefer each other over their current match. This algorithm would be perfect for the problem of matching students to advisors if only each advisor was accepting a single student but having the advisor able to have multiple students adds more complexity and so a slightly more complex algorithm must be used. While using the Stable Marriage is a good starting point, the kind of matching that would suit this problem would be better suited by the Student-Project Allocation System (SPA) (Abraham, Irving and Manlove, 2003) or by the Hospitals/Residents Problem (Roth and Peranson, 1999) which both allow many-to-one matching so supervisors can have multiple students until they reach their capacity.

In the project, a one-to-one match using something like SMP will not be suitable because supervisors will usually oversee multiple students. This is where something like HRP can be useful as it allows for many-to-one matching. It works using similar logic to SMP but in this example hospitals (in this case supervisors) can accept multiple residents (in this case students), up to a certain capacity. This kind of algorithm is used for things like medical residency matching and various other academic allocation systems where the stability of the matching process is essential. A benefit of using an algorithm like this is that it guarantees a stable outcome but it may not always produce a solution with the highest total satisfaction.

#### **Optimisation-Based Matching:**

An optimisation-based approach frames the allocation as an optimisation problem. This kind of algorithm has been used previously to try and solve a similar kind of problem which was researched and used simulated annealing (Chown, McCreesh and Prosser, 2018) to create matches. Each of the student's preferences are scored (e.g. first choice = 3 points, second = 2 points and third = 1 point) and the algorithm tries to maximise total preference satisfaction across all of the students. This process involves defining constraints such as each student is assigned only 1 project and that supervisors are not overloaded and maximising the score within these constraints. This process also allows for soft constraints to be built in such as having penalties to overall score if too many supervisor's projects go

unallocated. A benefit of using a process like this is that it can create a result that is the best match in terms of preference satisfaction but it may be difficult to explain why a student didn't get their first choice.

### **Linear Programming:**

Another approach that has been applied to similar allocations problems in linear programming (LP) which treats the problem as a maths optimisation model using an objective function, which could be something like maximising student satisfaction, and a set of constraints which would include things like supervisor capacity. This kind of approach has been used in similar allocation problems before like staff scheduling and course assignment (Şimşek et al., 2022) as this kind of approach is good at handling complex constraints. LP can require a lot more compute than greedy or stable marriage but will usually result in allocations that have a higher satisfaction outcome.

### **Greedy Matching:**

Using a greedy approach assigns each student one by one to their highest ranking available project. This means the system will iterate through each student and try to assign them their first choice. If their first choice is already at full capacity then it will move on to do the same for their second and third. If neither of those are available it would leave them unassigned and move onto the next student. The good thing about this approach is that it is quite simple so it can be understood by users which gives them an insight into why students were given their allocation. However using a greedy approach can lead to unfair matches due to the order in which students are processed has a big impact on how likely they are to get their first choice (Abraham, Irving and Manlove, 2003). The sooner the student is processed, the higher the likelihood they are to get their first choice.

There are a few existing tools that have been developed to assist in the process of matching students to projects. Some of them are available to purchase and others are open source. A few examples are:

- SPA-STUDENT (Abraham et al., 2003): This is an allocation system used by the University of Glasgow that uses stable marriage that involves both the preference of students and advisors for matching. However the limitations of this system is that it is fully integrated into the University's way of doing things so it would be difficult to adapt it to other situations.
- CAPR (Computer Assisted Project Allocation): This is an allocation framework that uses linear programming and adjustable preference weights to try and optimise matching. It's benefit is that it allows fine tuning of the preference weights which allows for increased student satisfaction and fairness. However, It is a more complex tool to use and would require lots of setup by someone who was a bit of a specialist with the tool so it wouldn't be so useful for any users that are new to using it. Similar optimisation approaches have been researched and used things like multi-objective binary programming (Şimşek et al., 2022) and genetic algorithms (Sánchez-Anguix et al., 2018) which both focus on identifying the trade-offs between optimisation and usability of the algorithms.

The project builds on what has already been researched about allocation algorithms and aims to merge it into something that is useful in practice for solving the issue of student allocation. Using the multiple algorithms to enable side by side analysis and visual comparisons, the app provides a tool that can be used by academic staff to make the process of allocating students to projects a more effective and time efficient task.

## **Specification**

This section focuses on the specific functionality that was intended for the app to have. Laying this out was useful for creating the design for the app since it made clear what features would be important to include. The first area to focus on would be the functional requirements which were identified when researching what things could be done to automate the allocation process.

#### **Functional Requirements:**

- Allow users to upload the data for student info, project info, supervisor info and pre allocated projects. The formats allowed should include CSV and Excel.
- Validate the input data for any issues such as missing data and duplicates.
- Use three different matching algorithms and be able to compare the results.
- Make sure that the pre allocated students are matched first and not changed by the algorithms.
- Display summary information about things such as number of matched students and satisfaction scores.
- Visualise the output match data using charts.
- Have users be able to export the allocations to Excel format.
- Allow modification of the default values for capacity.
- Warn users when the input data doesn't have enough capacity to match all students.
- Provide explanations to help users understand the visualisations and output data.

#### **Non-Functional Requirements:**

- Usability: The app should be easy to use for academic staff without technical background. The user interface should be intuitive and not require any programming knowledge to use.
- Performance: The app should be able to process data sets of at least 200 students without any noticeable delay and should not freeze or crash under any realistic data sets.
- Maintainability: The code should be well organised and commented. It should also be separated into clear sections for each area of functionality.
- Portability: The app must run on any of the modern browsers such as chrome or edge without any installation.
- Transparency: The reasoning behind the allocation should be understandable from the metrics and explanations provided.
- Security: The app doesn't use any storage or accounts but it should be resistant to any malicious uploads of anything that isn't csv or excel that aims to break the app

#### **Technology Stack:**

- The user interface framework that was chosen was Streamlit which allows for fast development of a web app using Python code. It supports things like file upload, charts and buttons.
- The programming language that was chosen was Python as it is suited to data manipulation and allocation algorithms and was one that the developer had previous experience with.
- The tool chosen to handle the optimisation algorithm was PuLP which is a linear programming solver that integrates well into Python.

- For handling the excel input and export, openpyxl was chosen as it allows for the reading and writing of excel sheets.
- For creating charts matplotlib was chosen for their clean look and easy integration into Streamlit.
- Streamlit Cloud was chosen for deploying the app so that users can access on the web without any downloads on whatever device they have.

Other alternative technologies were considered such as Flask as a web framework as it is quite flexible but has the downside of having to implement your own UI. React was also considered to create a JavaScript front end and then calling to a Python API for the data processing. It has the advantage of a lot more flexibility with the front end but comes at the cost of being a lot more complex to implement.

In terms of how the project was approached with regards to timeline, it was decided that the first few weeks would be spend doing some background research so that a good understanding of the problem and the available research and solutions that were out there. Once the background research was done it would be time to create a working prototype that could be tested by users and feedback gained. The development process ended up lasting the majority of the project but testing was done in time for some good improvements to be made to the app based on feedback. After the development process had run it's course it was time for fully writing up the report.

Throughout this period, meetings were had with the advisor when questions were had or any more information was required. It also provided a good opportunity for the advisor to give feedback about how the project is progressing. Minutes for these meetings can be found in Appendix 4. During the development stage, GitHub was used for version control and a project task-board was created to track all of the tasks and issues that needed to be solved. This was useful as it allowed for each development session to be focused on each GitHub issue and then once the issue was solved or new functionality added then it could be pushed to the repository and merged into the branch that provided the code for the hosted web app. An image of this project board can be found in Appendix 6.

## Design

This chapter covers the design choices that were made when developing the allocation system. The app was designed to be user friendly and effective at helping academic administrators allocate students fairly based on student preference and the set out constraints.

To give an overview of the app, it is web based and was built using Python and the Streamlit framework. It is intended to be used by academic staff who are tasked with the job of allocating students to projects. Using the app interface, the users can upload the data they have collected about student preference, the projects, supervisors, and any pre allocated projects, which will then be validated and put through the three different matching algorithms. They can then compare the results of the allocations using the charts and tables to decide which allocation they want to use from the downloadable excel workbook containing the final matches.

The structure of the app follows a modular design meaning that it has been broken into multiple components that do separate things. These components are:

- The frontend/UI: This component was built by integrating the whole code base into the Streamlit framework so that the code can be interacted with in the browser. Users use this component to upload



their data, change the configuration options and view the results.

- **Data Validation:** This component handles making sure that the input data is in the right format, that there are no missing required entries, and that there is sufficient capacity for a reasonably effective allocation to be made.
- **Matching Algorithms:** This component implements all three of the matching algorithms (Greedy, Stable Marriage, and Linear Programming) using the input data. Each of which has been separated into its own individual function.
- **Analysis and Visualisation:** This component calculates things like the satisfaction scores, project popularity and supervisor load then shows charts and tables to effectively display the outcomes of the allocation.
- **Export:** This component handles exporting the final allocations as an excel workbook using openpyxl so that the users can do further work on or distribute the allocations.

Having the app broken down into component parts was useful for testing and allows for future additions such as adding more functions to be a lot more manageable.

### Pipeline

This pipeline of each component doing what they are intended to do can be broken down even further so that it follows five stages which are laid out below. A flow chart of this pipeline can be viewed below in Figure 1.

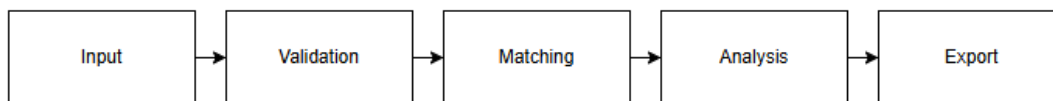


Figure1. Flow Chart of Pipeline

### Input Phase

- Users gather their collected data and upload the necessary files for the student preferences, supervisor information, project information, and pre allocated projects.
- The files can be either excel workbooks that contain all four sheets (An empty template can be downloaded from within the app) or individually uploading all four of the required csv files.
- The input data is then converted into pandas data frames which provide data structures for further processing to be done.

### Validation Phase

- Checks are done to ensure that the required columns are present, there aren't any missing values that are needed, there aren't any wrong data types and no duplicates of IDs.
- It then checks that all of the student preferences refer to a valid project that is contained in the project sheet.
- It makes sure that the supervisors capacities are numeric or that they fall back to the default capacity that has been set within the app.

- It displays the results of the validation as either a success or lists out any error messages that need to be dealt with before the app can proceed. Or it can also display any warnings about things like capacity that aren't going to stop the allocation process from being valid but might result in a sub par outcome.

### **Matching Phase**

- The pre allocated students are handles first before any of the allocation is done to make sure that they are set in stone before the algorithms are done because it was decided that pre allocated projects would take priority over preferred ones.
- Each algorithm is run separately but using the same input data that was handled in the input phase.
- A dictionary of each student ID to the project ID they have been assigned is then created for each of the algorithms.

### **Analysis Phase**

For each of the allocation results generated using the three algorithms:

- Satisfaction scores are computed using the weighting scheme of 3 points for 1st, 2 for 2nd, 3 for 3rd and 0 for unmatched.
- Charts and tables are created using the resulting allocation data.
- The supervisor load compared to their capacity is analysed.
- The most popular projects are identified based on how many times they were selected.

### **Export Phase**

- The dictionaries that were created by the matching algorithms are then merged with the corresponding data from the student, project and supervisor files so that all of the necessary information about the match can be contained on one excel sheet.
- The created excel workbook containing a sheet for each of the allocations created by the algorithms is then available to be downloaded using a button on the interface.

### **Matching Algorithms**

The main functionality of the app is the matching algorithms so it was important to get this part right. After researching the options that were available to be used to solve this kind of matching problem, three were chosen that seemed like they could do a good job. These being Greedy, Stable Marriage, and Linear Programming, each of which have their own benefits and trade offs. The inputs were kept consistent across all three implementations so that there could be a fair comparison.

#### **Greedy Matching**

Greedy allocation was chosen as it is similar to how the allocation would be done manually where the student assigned to their highest rank choice available. A simple pseudocode version of the algorithm can be found below.

```

For each student in random order:
  For each of their top 3 choices:
    If both project and supervisor have available capacity:
      Assign student to project
  
```

```
Decrease capacity counters
Break loop
```

The advantage of using a greedy algorithm is that it is simple, fast and easy to understand. However it can end up being a bit unfair based on the order of the input of students as the later the student is to get matched, the less likely they are to get a match they like. To reduce the chance of this unfairness, a randomiser was used to shuffle the order of students.

### **Stable Marriage**

This algorithm has been adapted from the Gale-Shapely matching algorithm and the implementation that is being used for this project is the context of the students proposing to their preferred projects. A simple pseudocode version of the algorithm can be found below.

```
While there are unmatched students:
    Student proposes to next project on their list
    If project (and supervisor) has capacity:
        Accept proposal
    Else:
        Reject and try next preference
```

The advantage of using this kind of algorithm is that it provides stability which means there is no pair of student and project that would rather be matched with each other than the match they have been assigned. However it can leave students unmatched if the capacity of projects are not sufficient. The algorithm has been adapted so that multiple students can be assigned to the same project and supervisor as long as the capacity isn't exceeded.

### **Linear Programming**

The linear programming algorithm created using PuLP tries to find the best way possible to allocate the students to projects under certain constraints. It does this by defining decision variables which are the yes or no choice for if a student gets a project, an objective function which is trying to maximise student satisfaction, adding constraints that must be followed and using a solver to search through all possible allocations and returning the best one. A simple pseudocode version of the algorithm can be found below.

```
Variables:
    x[s,p] = 1 if student s assigned to project p
Objective:
    Maximise total satisfaction score
Constraints:
    - Each student assigned to  $\leq 1$  project
    - Project and supervisor capacities not exceeded
    - Pre allocations enforced
```

An advantage of using this method is that it optimises student satisfaction and considers all constraints at the same time, however it can be slower on larger data sets.

### **User Interface**

The user interface was designed so that it would be easily useable by academic staff who do not have much technical training and still be effective.

Strategy for the layout:

- **Sidebar:** It was decided that having a collapsible sidebar that could be used for uploading files and configuring the capacities would be a good idea so that the user can use the sidebar at the start of the process then collapse it so that the analysis section that appears can take up the majority of the screen.
- **Main Panel:** This area takes up the rest of the screen and is used to display the tables and charts that can be used for analysis. It was also decided that the download button for exporting the final allocations would be at the bottom of the page so that the user could scroll through all of the analysis before downloading the results.

When thinking about the visual design it was decided that expandable sections would be a main feature of how the analysis would be displayed . It was decided that for each part of the analysis there would be a collapsible section for each algorithm that can be opened to see the analysis. This was decided because there is a lot of information on the page and it can become overwhelming when all of the sections are open at once so being able to hide the ones that were currently not relevant seemed like a good idea.

For deployment it was decided to go with Streamlit Cloud. It works locally but it requires some installation of packages to get things working so it was decided that it would be hosted on the web so that anyone with the link and access can run the app without any issues. After choosing to use the Streamlit framework for development it made sense to go with Streamlit Cloud as it was very fast and simple to get it up and running. The app also does not require any databases as all of the data is handled in memory during the session so there are no privacy concerns about data.

## Implementation and Testing

The implementation and testing phase of turning what was designed into a functioning app took up the majority of the time spent on the project. This chapter discusses the the decisions made behind the functionality, the features that were implemented, and how challenges faced during the process were overcome.

### System Overview

The app was created using Python as the core language with Streamlit providing the framework for turning that Python code into a web app. The way the app is structured can be separated into three broad areas:

- **Data Handling:** This includes reading the input data (Excel/CSV), validating that data to make sure there are no errors, transforming the data into a useable format for processing which in this case was pandas data frames.
- **Algorithms:** This involves implementing all three of the algorithms using the input data and outputting the resulting matches.
- **Web Display:** This part involves using Streamlit to give the user an interface to upload and download data, change the default capacities, and compare the results of the algorithms using the analysis sections provided.

## Data Handling

A challenge in the process of allocation is to make sure that the input data is valid and does not contain any errors. Without this validation process, things like duplicate IDs, missing fields and negative capacities could sneak into the matching algorithms and reduce the quality of the final allocations without the user noticing. This could cause students to not get as preferable a choice as they possibly could and would cause the user a lot more manual work to fix any issues that pop up later in the process.

The validation functions were grouped into three separate sections to improve clarity:

- Student: Checks that there aren't any duplicate IDs or choice, missing fields or invalid project IDs.
- Project: Makes sure each project has a unique ID and no negative capacities.
- Supervisor: Does the same checks as the student validation section but also includes a capacity check for invalid values.

Each of the validation functions returns error or warning messages to the user in the app using Streamlit. An example of a validation function for checking for duplicate student IDs can be found below in Figure 2.

```
# Check for duplicate IDs
if students_df['student_id'].duplicated().any():
    dup_ids = students_df[students_df['student_id'].duplicated()][['student_id']].tolist()
    errors.append(f"Duplicate student IDs: {dup_ids}")
```

Figure 2. Student Duplicate ID Validation - Lines 238-241

What the above validation code above does is checks through the student data frame and checks if there are any duplicate IDs. If there are then the duplicate student IDs are added to a list which is then added to the overall list of errors related to the student validation functions which are displayed to the user on the app using Streamlit.

Testing the validation was done using a dataset created that included deliberate errors called "test\_validation\_errors" which can be found with the rest of the test data in Appendix 1. Each error contained in the dataset was found and triggered the expected error messages. More information about this process can be found in the Evaluation section of the report.

## Matching Algorithms

The matching algorithms are the main functionality of the app and therefore the most amount of times was spend developing them and making sure they worked as intended. Each of the three algorithms have the same function but do it in a different way. They all take in the same data set of students and projects, apply their matching algorithm and output a data set of the allocations.

### Greedy Matching

Greedy is the simplest algorithm of the three, it iterates over the set of students once and tries to give each student the highest rank project that still has capacity. It has the benefit of being reasonably easy to explain because it is first come, first served while not going over capacity and is very fast so larger datasets will cause no issues of processing time.

All three of the algorithms start by locking in the pre allocated projects and make sure the capacities are adjusted accordingly before any of the matching is done. This code for this process can be viewed below in Figure 3.

```
# Start by assigning preallocated students
for _, row in preallocated_df.iterrows():
    sid, pid, sup = row['student_id'], row['project_id'], row['supervisor_id']
    allocation[sid] = pid
    supervisor_load[sup] += 1
    project_load[pid] += 1
```

Figure 3. Pre Allocated Matching - Lines 387-392

Once matching all of the pre allocated students is done it was decided that randomising the order of the students would be a good idea to try to improve fairness as Greedy matching causes students to have a higher chance of getting their preferred choice the closer they are to being the first student to be matched in the process. This randomiser can be viewed below in Figure 4.

```
# Randomize remaining students to avoid bias in order
students_df_shuffled = students_df.sample(frac=1).reset_index(drop=True)
```

Figure 4. Randomiser - Line 395

Greedy results in a fast output and is easier to explain to users but it can result in unfairness of allocation based on the order in which the students are selected to be matched.

### Stable Marriage Matching

The function that was created was inspired by the Gale-Shapley stable marriage algorithm but in a more simplified way as only one side (the students) have preference. Instead of taking the students in a random order it works more systematically through a process of four steps:

- Students are put in a queue and take turns proposing to their highest preferred project.
- If the project and its supervisor still have some capacity then the student is accepted.
- If not, then the student goes to the back of the queue and tries their next best choice when they get back to the front.
- This process continues until every student is matched or has run out of preferences.

In the code this is implemented using Python's deque data structure to manage the proposals which can be viewed below in Figure 5.

```
# Prepare a queue of students and their preference lists
student_prefs = {
    row['student_id']: deque([row['choice_1'], row['choice_2'], row['choice_3']])
    for _, row in students_df.iterrows() if row['student_id'] not in allocation
}
free_students = deque(student_prefs.keys())
```

Figure 5. Stable Marriage Queue - Lines 427-442

The loop then repeatedly pops a student from the queue, checks if their next preferred choice has availability and either assigns them or puts them at the back of the queue. This process in the code can be found below in Figure 6.

```

# Iteratively assign students to available projects
while free_students:
    sid = free_students.popleft()
    if not student_prefs[sid]:
        continue
    pid = student_prefs[sid].popleft()
    project_row = projects_df[projects_df['project_id'] == pid]
    if project_row.empty: continue
    sup = project_row['supervisor_id'].values[0]
    max_cap = project_capacity.get(pid)
    # If capacity is available, allocate; otherwise, requeue student
    if supervisor_load[sup] < supervisor_capacity[sup] and (max_cap is None or project_load[pid] < max_cap):
        allocation[sid] = pid
        supervisor_load[sup] += 1
        project_load[pid] += 1
    else:
        free_students.append(sid)

```

Figure 6. Stable Marriage Matching Algorithm - Lines 445-460

The main difference of this algorithm compared to the original Gale-Shapely is that projects do not have their own preferences to unmatch a student if a more preferred student proposes to them. So it is technically not a fully stable match but more of a proposal system. It could be modified to include project preference if that was something that was ever desired to be a feature in future.

The benefit of this kind of algorithm compared to Greedy is that it avoids some of the randomness contained in the Greedy match process as each student always gets a fair chance to cycle through all of their preferences. However a downside is that this method can result in more students being left unmatched when capacity is lower, this downside is explored further near the end of the evaluation section.

## Linear Programming

This approach to matching requires a lot more mathematics compared to the other two. Integer programming was done using the PuLP library which asks the solver to try and find the best possible set of allocations that maximises student satisfaction across the dataset.

In the code a binary decision variable is defined for each possible student to project pairing which can be viewed below in Figure 7.

```

# Create binary decision variables x[(student, project)]
x = LpVariable.dicts("assign", [(s, p) for s in students for p in projects], cat=LpBinary)

```

Figure 7. LP Binary Decision Variable Definition - Line 492

Four constraints are then added to ensure that:

- Each student gets at most one project:

```

# Constraint 1: Each student assigned to at most one project
for s in students:
    prob += lpSum(x[(s, p)] for p in projects) <= 1

```

- Pre allocated students are handled:

```
# Constraint 2: Respect preallocated students
for _, row in preallocated_df.iterrows():
    sid, pid = row['student_id'], row['project_id']
    for p in projects:
        prob += x[(sid, p)] == int(p == pid)
```

- Project and supervisor capacity is not exceeded:

```
# Constraint 3: Do not exceed project capacity
for p in projects:
    max_cap = project_capacity.get(p)
    if max_cap is not None:
        prob += lpSum(x[(s, p)] for s in students) <= max_cap

# Constraint 4: Do not exceed supervisor capacity
for sup, sup_cap in supervisor_capacity.items():
    sup_projects = [p for p, s in project_supervisors.items() if s == sup]
    prob += lpSum(x[(s, p)] for s in students for p in sup_projects) <= sup_cap
```

The satisfaction score is then added to the objective function so that a first choice gets 3 points, a second choice gets 2 points and a third choice gets 1 point. This can be viewed below in Figure 8.

```
# Objective: maximize sum of satisfaction scores
prob += lpSum(x[(s, p)] * student_choices.get(s, {}).get(p, 0) for s in students for p in projects)
```

Figure 8. LP Objective Function - Line 495

The solver then tries to find the solution that maximises the total score which can be viewed below in Figure 9.

```
# Solve optimization problem
prob.solve(PULP_CBC_CMD(msg=False))
```

Figure 9. LP Solver - Line 519

The solver will always find the best possible solution within the constraints that it is given and will not assign students to any projects outside their top three choices. In testing this algorithm resulted in the best average satisfaction score every time while also resulting in the lowest number of unallocated students.

## Web Display

The part that brings together the data handling and matching algorithms into a usable app is the Streamlit web interface. Using the Streamlit framework, a sidebar was created that contained the functionality for uploading data, adjusting the capacities and downloading the excel workbook that was designed to integrate with the app. Once the algorithms are run, all of the analysis sections will appear in dropdowns for the user to look at to help make comparisons. Graphs were created using matplotlib to give the user a more visual comparison of the graphs.

Another important feature that was implemented was the Excel export function which allows all of the allocations from each algorithm to be output in a single Excel workbook which contains all of the



necessary information for each allocation. Having this Excel workbook makes it easier for the user to make any further changes or to share with students or advisors.

Overall the implementation phase resulted in an app that has all the functionality that was set out for it to include which include three different matching algorithms, data validation, and a user friendly web interface.

## Evaluation

The main aims of the evaluation process were to make sure all of the app functionality worked as intended, to evaluate and compare the effectiveness of each algorithm and to evaluate how useful the tool was to a user of the app.

### Functionality Testing

Throughout the development process it was important to make sure that each of the functions were working as intended so there were not any unclear errors in later stages of the process. The first piece of major functionality that was tested was the data validation. Making sure that the user was made aware of any errors in the data set was important so that they did not follow through with the allocation process until they were fixed. The validation is split into three parts; student validation, project validation and supervisor validation.

#### Student Validation

The function for student validation has the following features:

- Checks all of the required columns for the sheet are present.
- Checks that there aren't any missing values in any of the required fields.
- Checks that there aren't any duplicate ID's.
- Checks that there aren't any invalid project ID's chosen.
- Checks that there aren't any duplicate choices of project.

#### Project Validation

The function for project validation has the following features:

- Checks all of the required columns for the sheet are present.
- Checks for duplicate project IDs.
- Checks for invalid capacity values (allows for empty but not negative numbers).

#### Supervisor Validation

- Checks all of the required columns for the sheet are present.
- Checks for duplicate supervisor ID's.
- Checks for invalid capacity values (allows for empty but not negative numbers).

All of the above checks are blocking checks which stop the app from moving forward to the allocation stage. In addition to those checks above that stop the process from moving forward, the app also gives the user warnings if any of the students have multiple choices in their top three from the same supervisor.

These warnings do not stop the app from moving to the allocation stage but could be useful for users if they wish to go back and make any changes to those particular students. This can be useful as making sure that all students have preference choices that are in line with the constraints to reduce the chance of the resulting allocations being unfair.

To test all of this functionality a made up data set called "test\_validation\_errors" (Which can be found in **Appendix ???** with the rest of the data sets) was created that contained errors to check that all parts of the above functionality were working as intended. This data set included missing required columns, null values, duplicate IDs, invalid project ID references and negative capacities. In all of the mentioned cases the app correctly identifies the errors and displays them to the user and stops the app from moving on to the allocation process. An example of the output that will be produced when invalid data is loaded in can be found below in Figure 10.

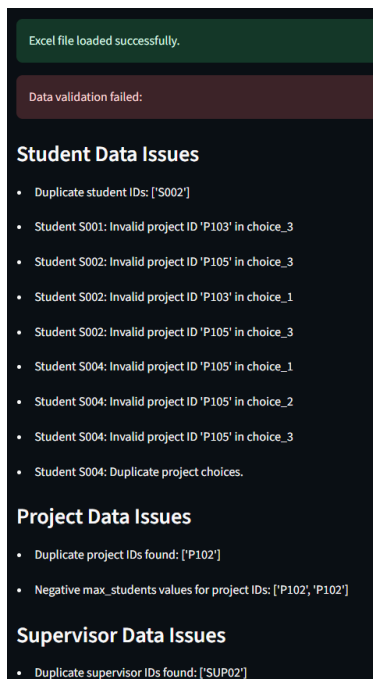


Figure 10. Data Validation Errors

### Algorithm Effectiveness

Once the validation was found to be functioning as intended, the next thing to evaluate was to compare the matching algorithms using made up and anonymised real data sets. Three different algorithms were used in the app which were mentioned earlier; Greedy matching, Stable Marriage matching and Linear Programming. Each algorithm was tested using the same data sets so that the outcomes of the matches could be compared fairly.

Multiple ways of comparing the algorithms were used to get a broad range of comparison. These methods include:

- Percentage of first choice allocations: which links to student happiness at the individual level.
- Overall satisfaction score: Which is the overall fairness across the set of students.
- Choice Distribution: Which increases insight into the allocations.

- Unallocated Students: Which relates to workload reduction for users.

### First Choice Percentage

The percentage of students allocated to their first choice was one good indicator that the algorithm had done a good job so this was an important metric to measure and compare. The two made up data sets that had more realistic data and the anonymised previous years data was used to make a comparison. You can view how the first choice percentage varied for each algorithm in the different data sets in the table below.

Data Set	Greedy	Stable Marriage	LP
"combined_input"	50%	79%	81%
"project_data_sanitized"	86%	95%	76%
"realistic_input_data_50_students"	78%	86%	80%

The performance does vary across the datasets and algorithms. Greedy matching got a high percentage in two of the cases but it was less consistent overall which can be seen in the table where it drops to 50%. Stable Marriage achieved consistently good results which shows that stability of the matches doesn't always have to cost getting more first choice allocations. Linear Programming had okay results as it didn't do as well in some of the data sets but did better in others. This reflects the fact that LP prioritises overall satisfaction which can mean trading off a few first choices in order to improve the whole set of allocations. In the example of "project\_data\_sanitized", LP had the lower first choice percentage compared to Stable Marriage but it had 6 extra matches, 11 second place matches compared to Stable Marriage which only had 2 and had the higher average score of 2.7 compared to Stable Marriage which had 2.5. This shows that while the first choice percentage is a good indicator it does bias towards the other two algorithms compared to LP.

### Overall Satisfaction Score

The overall satisfaction score is another good indicator of how well the algorithms performed. A similar comparison was done to the previous first choice percentage of comparing multiple data sets to see how the average satisfaction scores compared. The results can be found in the table below. The average satisfaction score is calculated by giving 3 points to a 1st choice, 2 points to a 2nd choice, 1 point to a 3rd choice and 0 points to any unmatched students. The average of all of the points is then taken as the overall satisfaction score of the algorithm. The table below shows the average satisfaction score for each algorithm.

Data Set	Greedy	Stable Marriage	LP
"project_data_sanitized"	2.53	2.51	2.7
"realistic_input_data_50_students"	2.22	2.36	2.48
"combined_input"	0.43	0.59	0.65

The results in the table above show a clearer difference in the algorithms than the first choice percentage. While greedy seems to have done well in the first two data sets, when presented with a more challenging data set of "combined\_input" which has far too many students for the available capacity, it doesn't do as well as the other two algorithms. This highlights that the Greedy algorithm might not be the best choice to go with if the data set is not ideal. Stable Marriage did well in all three datasets but was consistently beaten by Linear Programming which had the highest overall satisfaction in every dataset that has been tested. Due to the way the LP algorithm works, it's optimisation allows it to assign more students to their

2nd or 3rd choices rather than leaving them unassigned. These results suggest that while 1st choice percentage can make Stable Marriage seem like a better choice, the average satisfaction score shows that LP is more effective at producing allocations that maximise overall satisfaction.

### Choice Distribution

1st choice percentage and average satisfaction are useful metrics for figuring out which algorithm might be better to use but they can sometimes hide how the allocations are spread across each preference. A pair of stacked bar charts can be viewed below which show the choice distribution for each of the algorithms when the dataset "project\_data\_sanitized" and "realistic\_input\_data\_50\_students" were used.

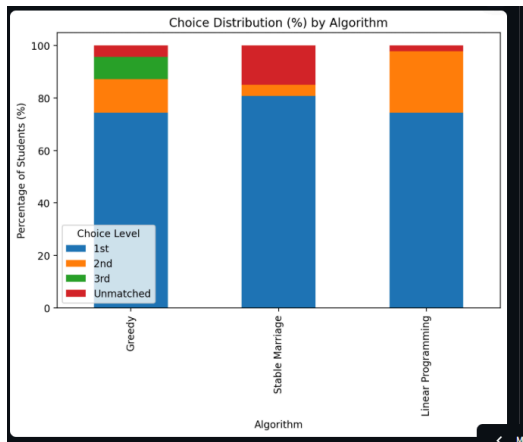


Figure 11. "project\_data\_sanitized" Choice Distribution

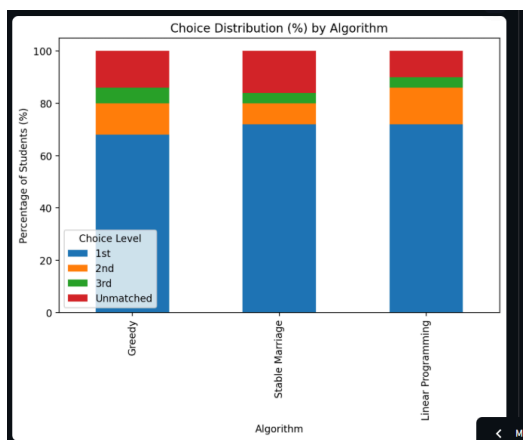


Figure 12. "realistic\_input\_data\_50\_students" Choice Distribution

What can be seen from both of the charts is that LP will usually result in the least number of unallocated students. It also results consistently in a larger percentage of higher choices. This can especially be seen in the first graph in the comparison between LP and Stable Marriage where LP has sacrificed a number of 1st choices to reduce the number of unallocated students by increasing the number of 2nd choice students. This is a clearer visualiser of what was mentioned in the previous overall satisfaction score section.

### Unallocated Students

Comparing the number of resulting unallocated students from each algorithm as this could be a big factor for users deciding which algorithm they want to go with. The table below shows the number of unallocated students for each of the algorithm using the same three datasets as before.

Data Set	Greedy	Stable Marriage	LP
"project_data_sanitized"	4	7	1
"realistic_input_data_50_students"	7	8	5
"combined_input"	76	76	74

The number of unallocated students may represent increased dissatisfaction and will require the user to then go on and do more manual work to solve the issue of unallocated students. LP had the lowest number of unallocated students in all datasets tested and Stable Marriage tends to leave more students unallocated compared to the other two algorithms.

## User Testing

To try and get an understanding of how useful and intuitive the app was for users to use it was given to an advisor who had experience manually assigning students to projects to test how useful it would be. They were given the same set of data sets that were mentioned and which can be found in the appendix. The feedback highlighted some main points:

- Having configurable default capacities: The prototype that was tested had a hard coded capacity of 3 to fall back on if the capacities were missing in the input file. Being able to change that within the app was something that was mentioned and was then a feature that was added which will be explored further in the next chapter.
- Adding a single chart to compare all 3 algorithms: This was suggested so that it would be easier to get a visual idea of how the algorithms compared. This suggestion led to the creation of the choice distribution comparison graph and average satisfaction score comparison graph which are both shown at the end of the next chapter.
- Adding explanation for what satisfaction meant: It wasn't so clear to the user what was meant by the satisfaction score so an explanation was added above the satisfaction analysis section which explained how the scoring system worked.

Further testing of more users would have been useful to get a broader range of feedback so more improvements could have been identified but great suggestions were received and acted upon.

Overall, the evaluation process showed that the app works as intended. The validation features work reliably which prevents datasets with errors from being used for allocations and that any issues are made known to the user. Using the test datasets to compare algorithms showed that Stable Marriage and Greedy matching both have strengths in some areas but LP seems to have produced the most reliable results in maximising satisfaction and minimising unallocated students.

## Description of the Final Product

The final product is a web app that helps streamline the process of allocating students to projects. This chapter will focus on the functionality and features of the final product.

When the application is launched, which can be done either by opening the hosted app on Streamlit for those who have the link and permissions or using the command line prompt provided at the top of the codebase, the user will be met with a clean interface which is split into the sidebar on the left and a main content panel which takes up the rest of the space. The main interface when first opened can be viewed below in Figure 13.

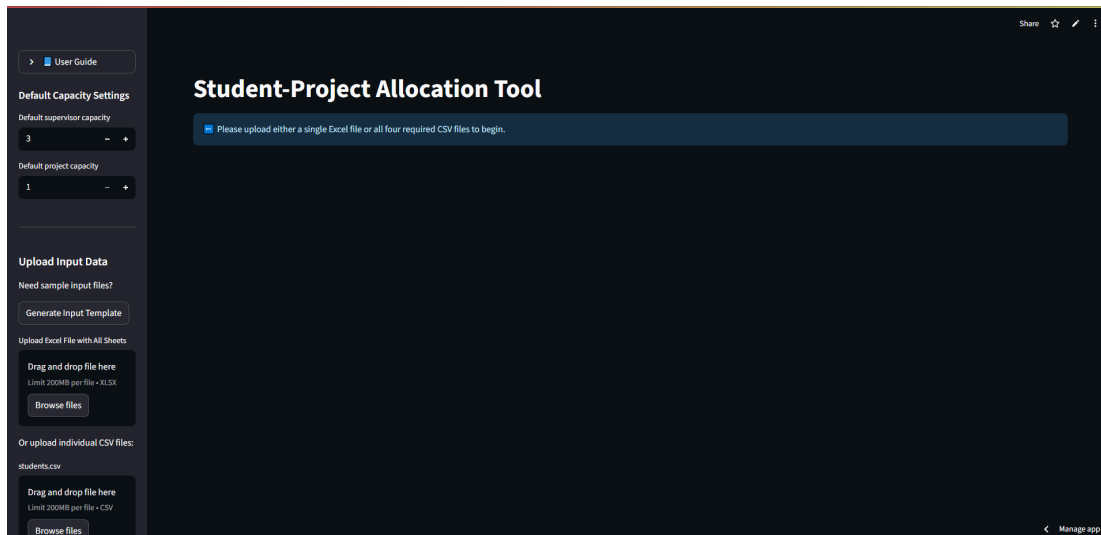


Figure 13. Main UI

## Sidebar

Figure 13. Shows the sidebar on the left and it contains the functionality for the help menu, capacity settings, generating the empty excel template, uploading a complete excel workbook and uploading CSV files.

At the top of the sidebar there is a dropdown that contains a brief user guide of how to use the application. It doesn't go into great detail but is enough to get the user started. This user guide can be opened and closed by pressing the button at the top. This user guide can be viewed below in Figure 14.

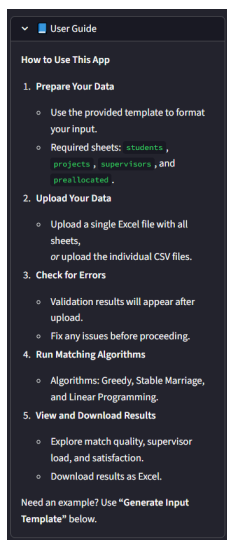
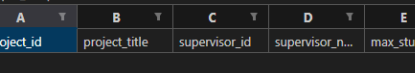


Figure 14. Open User Guide

The next feature of the sidebar is the configuration settings. There is the option to change the default values for both the supervisor and project capacity for when a capacity is not contained within the input data. These capacities can be changed by either typing in a different number into the text field or increased or decreased by increments of one using the plus or minus button, all of which can be viewed in Figure 13.

The feature below the configuration settings is the Excel template download button. If the user has not yet collected their data set yet or it is in a format that does not integrate with the app, they can download an empty excel workbook which contains all of the necessary and useful fields that are required for matching. All the user needs to do is press the button and it will download the excel template. The excel template can be viewed below in Figure 15 where you can see there is a sheet for student, project, advisor and pre allocated student data.



input\_template.xlsx

	A	B	C	D	E
1	project_id	project_title	supervisor_id	supervisor_n...	max_students

students projects supervisors preallocated

Figure 15. Empty Excel Template

The feature that comes next in the sidebar is the input buttons to upload the data sets that are to be put through the matching algorithms. This section is split into two parts; the first part being the button to upload an Excel workbook in the same format as the downloadable template, the other part being four different buttons to upload the CSV files for students, projects, supervisors and pre allocated. When the buttons are pressed it opens up the file explorer on the device being used and the files can be selected and they will be uploaded into the app to start the validation process. The mentioned buttons can be viewed at the bottom of the sidebar in Figure 13.

## Main Content Area

Once the data sets have been uploaded using the buttons in the sidebar the main area on the right that takes up most of the app will start populating with information. An example of which can be viewed below in Figure 16.

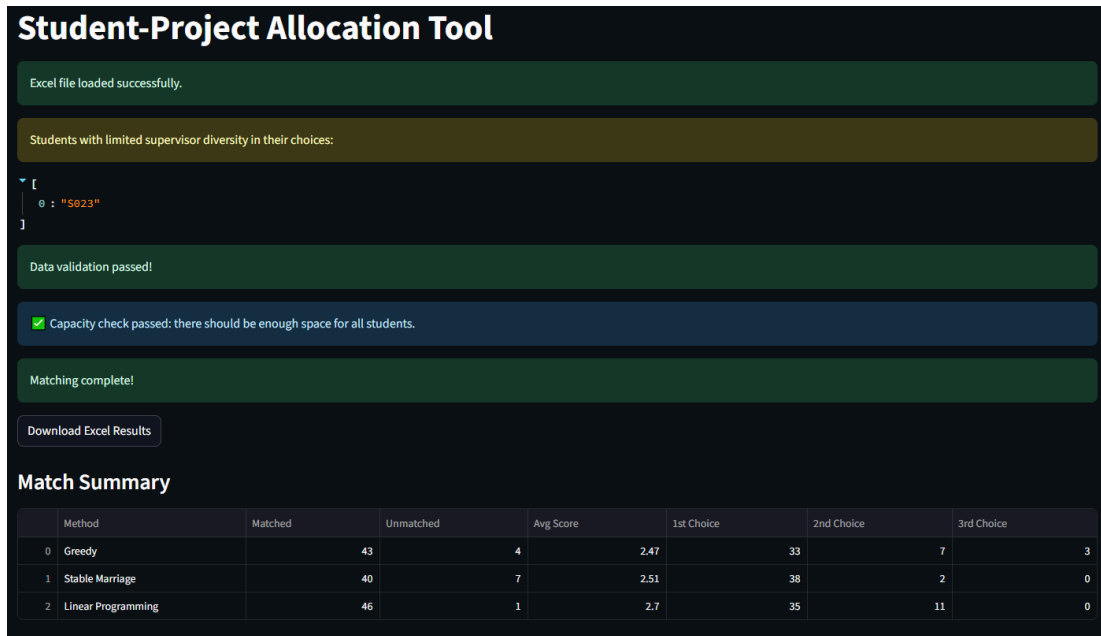


Figure 16. UI With Loaded Data

### Validation

If the data has been successfully loaded then it will show a message at the top of the screen which has a green background saying that the file has loaded successfully. It will then run all of the data validation functions on the input data and will show any warnings or suggestions. In Figure 16. above, it can be seen that there is an issue with a student who has the same advisor as more than one of their choices. This isn't an issue which will stop the matching process but is something that could be useful for the user to know if they want to make any changes to that students preferences before continuing. If there are more serious issues with the data that are found in the validation process then it will list off all of the issues without going forward with the matching. An example of which can be viewed below in Figure 17.



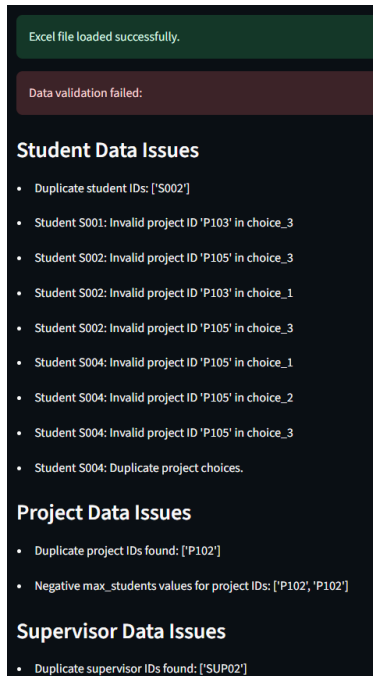


Figure 17. Validation Errors

If the validation is passed then it will show a message with a green background saying validation passed which can be viewed in Figure 16. The app will then do a capacity check which just makes sure that based on the capacity values that there is actually enough capacity for every student to be matched. If there is not enough capacity then it will show an error message that can be viewed below in Figure 18. A capacity warning will not stop the allocation process from going forward but does give the user a warning so they can decide if they want to increase supervisor capacity before doing the allocation.

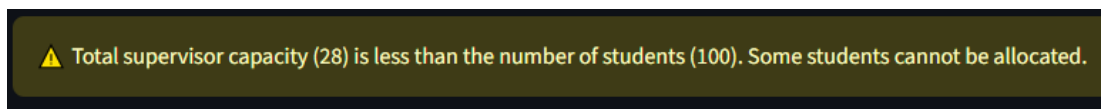


Figure 18. Capacity Issue

## Exporting

Once the matching process is complete it will show a message and a button to export the results will be found just below which can be viewed in Figure 4. When the download excel results button is pressed it will download an excel workbook that contains all of the allocation results for each of the algorithms. There is a page for each algorithm the structure of the pages can be viewed below in Figure 19.

	A	B	C	D	E	F	G
	T	T	T	T	T	T	T
1	student_id	student_name	assigned_project_id	assigned_project_name	supervisor_name	supervisor_email	assigned_choice
2	S001	Ava Lambert	P016	Cloud-Based Medical Report Organizer	Dr Iain Martin		2nd
3	S002	Felix Moreno	P036	Automated Plagiarism Checker	Dr Ricky Qiu		3rd
4	S003	Jasmine Patel	P057	Natural Disaster Alert System Using Sensors	Dr Gemma Webster		1st
5	S004	Kai Nakamura	P054	Climate Data Visualization Platform	Mr Oluwafemi Samuel		1st
6	S005	Isla Briggs	P025	Interactive SQL Learning Game	Dr Rachel Menzies		1st
7	S006	Mateo Alvarez	P004	Real-Time Language Translation App	Dr Michael Crabb		1st
8	S007	Priya Desai	P043	Voice-Controlled Personal Assistant	Dr Craig Ramsay		1st
9	S008	Leo Thornton	P059	Multi-Language Text Summarization Tool	Dr Gemma Webster		1st
10	S009	Amara Singh	P001	AI-Powered Personal Finance Tracker	Prof. John Annett		1st
11	S010	Ezra Cohen	P013	Online Exam Proctoring Using ML	Dr Ludovic Magerand		2nd
12	S011	Lucia Moretti	P031	AI Tutor for Python Programming	Dr Brian Plass		1st
13	S012	Hugo Reyes	P062	Smart Reminder Bot for Daily Routines	Prof. Annalu Waller		1st
14	S013	Zoe Lin	P005	IoT-Enabled Smart Farming Dashboard	Dr Michael Crabb		1st
15	S014	Oliver Maddox	P029	Digital Library with Recommendation Engine	Dr Iain Murray		1st
16	S015	Ayla Karim	P040	Digital Expense Splitter for Groups	Dr Craig Ramsay		1st
17	S016	Ethan Brooks	P013	Online Exam Proctoring Using ML	Dr Ludovic Magerand		1st
18	S017	Mirela Novak	UNASSIGNED			Unknown	Unassigned
19	S018	Noah Schroeder	UNASSIGNED			Unknown	Unassigned
20	S019	Yara Haddad	P050	AI-Based Code Refactoring Tool	Dr Daniel Rough		1st
21	S020	Liam Price	P007	AI Chatbot for Mental Health Support	Dr Michael Crabb		1st
22	S021	Sofia Kim	P033	IoT-Based Home Security System	Dr Brian Plass		1st
23	S022	Idris Mason	P032	Real-Time Crime Map Visualization	Dr Brian Plass		1st
24	S023	Naomi Feldman	UNASSIGNED			Unknown	Unassigned
25	S024	Theo Zhang	P017	Gamified Learning Platform for Coding	Dr Iain Martin		1st
26	S025	Keira O'Donnell	P012	Smart Waste Management System	Dr Ludovic Magerand		2nd
27	S026	Milo Santiago	P019	Secure File Sharing via Blockchain	Dr Iain Martin		1st
28	S027	Saanvi Reddy	P007	AI Chatbot for Mental Health Support	Dr Michael Crabb		2nd
29	S028	Callum Fraser	P066	Live Lecture Transcription Tool	Prof. Annalu Waller		1st

Figure 19. Exported Excel Workbook Containing Allocations

## Analysis

The next thing that will be shown on the page is the match summary table. This contains information about the results of each matching algorithm. It lets the user know how many students were matched or left unmatched, how many students got their first, second or third choice and the average score. This table can be viewed at the bottom of Figure 16.

As the page is scrolled down the user is met with the analysis sections which contains three expandable sections for each algorithm for each part of analysis which can be viewed below in Figure 20. Each of these can be opened to view the content and closed to that it doesn't clutter the screen

Analysis
> View Greedy Matching Analysis
> View Stable Marriage Analysis
> View Linear Programming Analysis

Satisfaction
Definition: Each student is awarded 3 points if matched to their 1st choice, 2 points for 2nd choice, 1 point for 3rd choice, and 0 points otherwise.
> Satisfaction - Greedy Matching
> Satisfaction - Stable Marriage
> Satisfaction - Linear Programming

Supervisor Load Analysis
> Greedy Matching - Supervisor Load
> Stable Marriage - Supervisor Load
> Linear Programming - Supervisor Load

Project Popularity & Utilization
> Greedy Matching - Project Popularity

Figure 20. Analysis Section Expandable

The first section of analysis is the matching analysis which contains a table of choice preference distribution. This table shows how many students got their first, second or third choice, and how many were matched or unmatched. It also has a supervisor load distribution table which has each of the supervisors and how many students were assigned to them. It then will list off the names of any students

that have been assigned to projects that are outside of their top 3 choices and has a table showing how many students were assigned to each project. This can all be viewed in Figure 21. There are three of these sections which are all the same structure but are populated by the data of each of the three different algorithms.

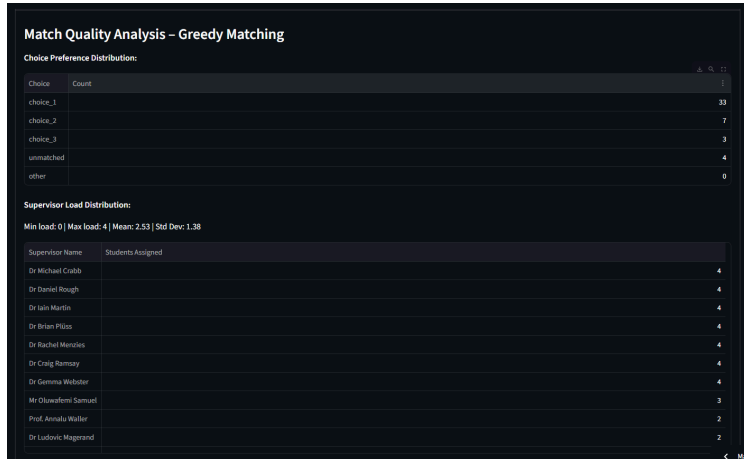


Figure 21. Match Analysis Section

The next section of analysis is the satisfaction scores. This section contains a graph for each algorithm showing how many students got their first, second and third choice to give a more visual explanation of the results compared to the table. An example of which can be viewed in Figure 22. The satisfaction score is calculated by giving three points if a student is matched to their first choice, two if matched to their second, and one if they are matched to their third choice.

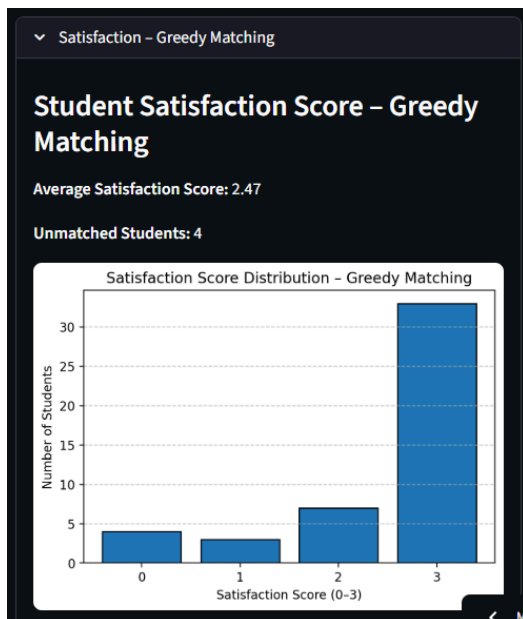


Figure 22. Satisfaction Score Graph

The next section of analysis is a more in depth supervisor load analysis. This contains a graph that shows each supervisor and their number of assigned students compared to their capacity. This gives the user an idea of which supervisors were at their capacity and which were under assigned. An example of which can be viewed below in Figure 23.

**Supervisor Load Analysis**

Greedy Matching - Supervisor Load

**Supervisor Load Analysis - Greedy**

	Supervisor Name	Supervisor ID	Assigned	Capacity	Status
0	Prof. John Amott	SUP01	1	1	OPTIMAL
1	Dr Michael Crobb	SUP02	4	4	OPTIMAL
2	Dr Vladimir Jancic	SUP03	1	1	OPTIMAL
3	Dr Ludovic Wagerand	SUP04	2	4	UNDERUSED
4	Dr Iain Martin	SUP05	4	4	OPTIMAL
5	Prof. Stephen McKenna	SUP06	1	1	OPTIMAL
6	Dr Rachel Mansles	SUP07	4	4	OPTIMAL
7	Dr Iain Murray	SUP08	2	2	OPTIMAL
8	Dr Brian Piloss	SUP09	4	4	OPTIMAL
9	Dr Ricky Qiu	SUP10	2	2	OPTIMAL

Figure 23. Supervisor Load Analysis

The next section of analysis is the project popularity analysis table. This table shows each of the table and the number of requests for that project compared to the number of times it was assigned. This gives the user an idea of which projects were popular and which were not so they could shift supervisor capacities around to try and give more students their preferred choice that if that was something they desired. There is a table for each algorithm just as for each of the previous analysis sections which can be viewed below in Figure 24.

**Project Popularity & Utilization**

Greedy Matching - Project Popularity

**Project Popularity & Utilization - Greedy**

Project Title	Requested	Assigned	Status
Online Exam Proctoring Using ML	4	2	UNDER-ASSIGNED
Collaborative Code Review Platform	3	2	UNDER-ASSIGNED
AI-Based Code Refactoring Tool	3	2	UNDER-ASSIGNED
Smart Attendance System Using Face Recognition	4	1	UNDER-ASSIGNED
AI-Powered Personal Finance Tracker	4	1	UNDER-ASSIGNED
Online Resume Analyzer with Suggestions	1	1	MATCHED
IoT Enabled Smart Farming Dashboard	2	1	UNDER-ASSIGNED
Real-Time Language Translation App	3	1	UNDER-ASSIGNED
Virtual Reality Chemistry Lab Simulator	3	1	UNDER-ASSIGNED
AI-Based Resume Evaluator	3	1	UNDER-ASSIGNED

Figure 24. Project Popularity Table

The last two pieces of analysis are both graphs. The first of which is a choice distribution comparison stacked bar chart which compares the percentage of students that got each of their top three choices for each algorithm against each other. This chart can be found below in Figure 25.

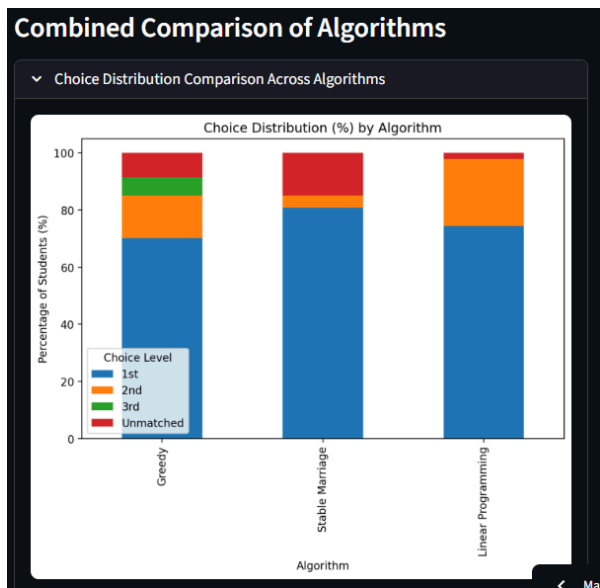


Figure 25. Choice Distribution Comparison Graph

The other graph is one that compares the average satisfaction scores of each of the algorithms which can be found below in Figure 26.

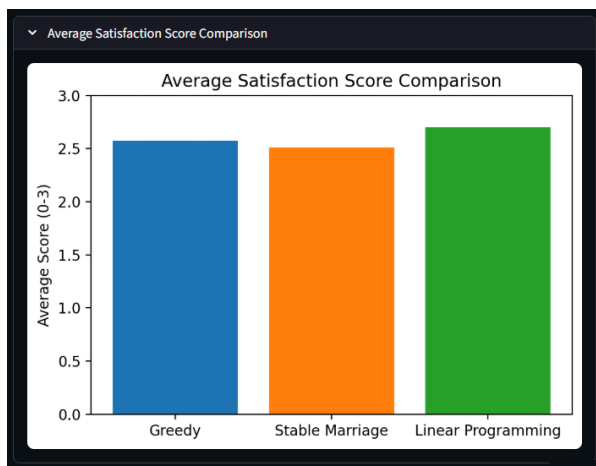


Figure 26. Average Satisfaction Comparison Graph

## Appraisal

This chapter reflects of how the project of creating the allocation system went as a whole. It highlights what went well, what could have been done differently, what advice would be given to others doing a similar project, and what lessons were learned looking back over the project.

One thing that went well was the selection of technologies to use to create the app. Using Python and Streamlit allowed for a prototype of the app to be created very early on due to it removing the need to

spend time and focus on creating functionality for things like file uploading, charts and expanders. If doing the project again Streamlit would definitely still be the top choice of frameworks to use.

Another thing that went well was the inclusion of the three matching algorithms which gives value to the users by allowing for comparison so they can choose which result best suits their situation. Looking back it was definitely a good decision to include multiple algorithms rather than just focusing on trying to make one work as best as possible.

One thing that could have gone better would have been to include more real world testing of the app. The majority of the testing was done using made up data sets and an anonymised data set from a previous years allocation process and was only tested by one user which limited feedback for improvements. The testing using the made up data sets was very useful but looking back if the project was to be done again it would be a focus to get a working prototype ready earlier and have multiple users test data sets to get a wider range of feedback.

Another thing that could have been done better would have been to get a draft of the final report done sooner. Beginning to write up the majority of the report was left very late in the timeline of the project which left little time for the report to be read over by the advisor and any suggested changes to be made. Having already had a good chunk of the report ready to be evaluated and improved upon would have allowed for less of a rush at the end of the project to get the report done which will have likely reduced the quality of the report to be lower that in could have been.

Looking back on the project there were a few lessons learned. One of these being the importance of keeping the code modular. Because it was decided early on that the code would be split up into different sections that would do their own individual tasks it made it far more easy to add new functionality to the app without breaking or causing any errors in the other parts. This allowed for a lot of time to be saved from not having to spend hours trying to debug why certain issues were occurring. It also meant that when changing the initial prototype from just python code into using the Streamlit framework it was easy to modify each component.

Another lesson that was learned was to try and engage with users sooner to get feedback. It was quite late in the project timeline that any feedback on the app was received which left minimal time to spend on making any significant changes or improvements. Although some great improvements were made after testing was done by the advisor, any major adjustments of functionality or how the app worked were off the table as there wouldn't have been enough time to implement them. Having had more feedback on the app may have resulted in a more effective user interface and a more streamlined process of allocation.

To evaluate the project as a whole, it did succeed in meeting the main goals that were set out. A working app was created that allocates students to projects based on their preferences using multiple different algorithms, it integrates data validation and analysis of the results into the app and allows for the results to be exported into a format that will be of use to the users.

## Summary and Conclusions

The project set out to design, implement, and evaluate a web application that helps with a regular challenge for academic administrators of assigning students to their final year projects. The main motivations for this project was to reduce the pain of manual allocation processes that can be time

consuming for staff and to improve the fairness and effectiveness of the allocation process so that more students end up with their more preferred choice.

The result of this project is a fully functional web app built using Python and the Streamlit framework. It has been designed with a user friendly interface that guides staff through the allocation process. The app allows for the user to input student, project, supervisor and pre allocated projects, validates the data, uses three different matching algorithms (Greedy, Stable Marriage, and Linear Programming), and outputs the matches to an excel spreadsheet that can be helpful for the user.

A significant goal that was achieved during the project was to get three different and modular matching algorithms working so that they could be compared and an appropriate set of assignments chosen. Testing showed that the Linear Programming algorithm consistently resulted in the highest satisfaction score for students and more students were usually assigned to their first choice while also having fewer unassigned students overall. However, the Greedy and Stable Marriage functions still offered value by being a good option for when constraints require simplicity and speed to be a priority.

Another major success of the project was creating a robust validation layer before the allocations were done. This was important to get right because it minimises any manual errors that could cause students to not get their best choices. This increases the fairness of the project and gives students more confidence that they are getting a fair shot at being assigned to one of their preferred projects.

The analysis section was also an area of worth to note as it allows the users to have better insight into the fairness and effectiveness of the allocation process that is being automated. Having charts, tables and metrics allows the users to understand the final allocations and how each algorithm reaches it's results.

The app was evaluated using both made up data and an anonymised set of data from a previous years allocation process. A key finding was that the Linear Programming algorithm resulted in the highest satisfaction score when tested on all of the data sets. Another key finding was that Greedy matching, while sometimes effective and fast, would regularly leave students unmatched or assigned to one of their lower preference choice. The app was able to handle data sets of around 100 students without any noticeable slowing of the app which indicates it will work just fine for any common data set that the app would be used on.

From a development view, this project provided many opportunities to learn new skills across a few different areas such as:

- Algorithms: Learning about and implementing matching algorithms allowed for insight to be gained about optimisation and fairness.
- Data Processing: Learning to use Pandas and making sure that the data was valid and workable was a key part of the development process.
- Web Development: Using Streamlit to create a web interface that could be understood and easily used by academic staff.

This project delivered a practical and effective tool for automating the problem of allocating students to projects. It allows for academic staff who are not familiar with algorithms to effectively use them in the allocation process. While there is room for further improvements, this project has created a strong foundation as a prototype and makes clear that algorithms can be of great use when integrated into systems to solve real world challenges.

## Recommendations for Future Work

The current app has all of the features that were intended to be implemented and provides a useful tool but there are still some areas that could be improved or expanded. This project was focused primarily on the matching of the students to projects but there are also many other areas of the whole process that could be improved and added to the app.

One area of improvement could be to integrate the process of gathering the student choices and information. The app as it currently is requires a data set of student choices that manually have to be gathered and collated by the user before they can run the allocations. Automating the sending out of forms and collation of student data that would integrate into the allocation process automatically would be a great addition to the app and the whole allocation process. The app could also be integrated into whatever system the university is using to store student and supervisor data so that they can be integrated without manual data collection. There could also be the addition of having an option to submit a final allocation excel sheet that would be processed and send out emails to all of the students and advisors making them aware of their allocations.

Another area of improvement could be to include supervisor preference. The supervisors could also have a preference rank for students. This could allow for a better overall experience not only for students but for the advisors as the pairing would end up so that both have a more preferred choice of who they are working with. This could involve altering the stable marriage algorithm to be more similar to the original Gale-Shapely algorithm which would consider the preference of both sides.

Another potential area of improvement could be the addition of supporting group projects where multiple students have input into which project they are doing. It is common that final year projects are individual but the app could be widened to support the allocation of group projects which are a common occurrence in universities. This could involve having groups formed based on their project selection and things like projects having multiple advisors.

In summary, the app currently provides great value and a solid foundation for individual student allocation to projects but there is room for improvements to be made. The app could be developed to be even more specific for departments of the university or could be broadened to be applicable in a wider range of allocation circumstances.

## References

- Gale, D. and Shapley, L.S. (1962) 'College admissions and the stability of marriage'. Available at: <https://wayback.archive-it.org/5456/20240920170021/http://www.eecs.harvard.edu/cs286r/courses/fall09/papers/galeshapley.pdf>
- Abraham, D.J., Irving, R.W. and Manlove, D.F. (2003) 'The student-project allocation problem'. Available at: <https://www.cs.cmu.edu/~dabraham/papers/aim04.pdf>
- Chown, J., McCreesh, C. and Prosser, P. (2018) 'Approaches to student-project allocation'. Available at: <https://arxiv.org/pdf/1810.11370>



Ayegba, P., Olaosebikan, S. and Manlove, D.F. (2025) 'Structural properties of the student–project allocation problem'. Available at: <https://arxiv.org/pdf/2501.18343>

Sánchez-Anguix, V., Julián, V., Palomares, A. and Botti, V. (2018) 'On the use of genetic algorithms to solve the student–project allocation problem'. Available at: <https://arxiv.org/pdf/1812.06474>

Şimşek, A., Ozdemir, Y. and Karsu, Ö. (2022) 'A multi-objective binary programming approach for student–project allocation'. Available at:  
<https://www.sciencedirect.com/science/article/abs/pii/S095741742101407X>

Roth, A.E. and Peranson, E. (1999) 'The redesign of the matching market for American physicians: Some engineering aspects of economic design'. Available at:  
<https://natmatch.com/documents/rothperansonaer.pdf>

## Appendices

Appendix 1: Source Code and Testing Files

Appendix 2: Risk Assessment Form

Appendix 3: Ethics Declaration

Appendix 4: Advisor Meeting Minutes

Appendix 5: User Guide

Appendix 6: GitHub