

Student Dashboard Management (SDM)

Name : Warren Mao

Github URL : <https://github.com/Warrenmao1/Y12-AT3>

Student Marks Program

Add Subject

English

Add name & mark

Show frequency histogram

Show individual marks

Calculate Performance

Identifying and Defining

1.1 Problem Statement

Many analytical systems used for judging student academics may overcomplicate data, using a range of indicators. This may create confusion or misunderstanding for those without a very strong background in statistics, generating inconvenience for those such as teachers plotting data or for audiences such as parents trying to understand it.

1.2 Project Purpose and Boundaries

I plan to create a program that allows teachers to easily monitor student performance through first inputting subjects, then student grades for that subject. The program would then automatically graph the data whilst providing a summary of each student's performance. This would provide ease for teachers by automating graphing features and calculation features whilst summarising it in appealing graphics. However, it should be noted that predicting marks is beyond the boundaries of this project, with machine learning and incorporation being out of the scope due to concerns of its reliability.

1.3 Stakeholder Requirements

The stakeholders of this project are education institutions that require a modernised analytics system. These investors require the SDM to balance providing deep analytics whilst also summarizing this information in a clear communicative way. Additionally, the system should have a way of securely storing user data and information for significant organisations such as the department of education.

1.4 Functional and Non - Functional Requirements

- Functional
 - A user password system. It should allow the teacher to enter a username corresponding with a password which is either stored when signing up or checked if it matches one another for logging in.
 - The ability for students' names and marks inputted across multiple subjects of the user's choosing.
 - A calculation algorithm to determine the mean and standard deviation of student grades. It should compare students' grades for their respective subject to the mean through the use of the Z score.
 - An external UI from the console which allows users to type and input marks and display a frequency histogram.
- Non functional
 - The sign up page should ensure the username is not the same as one existing in the database. Both the login and signup system should ensure a password and

username are entered at all in the text boxes. Additionally hashing is to be incorporated for passwords

- In each mark text box for each student, multiple scores should be allowed to be entered with an average being calculated from these scores.
- The UI should show a very clear and readable curve, using colours to symbolize student performance.
- Error handling is to be incorporated to check inputs that match a criteria such as marks being in the form of numbers rather than letter grades and provide a succinct error message of the reason for error.

1.5 Constraints

The main constraints on this project is the time frame window and implementing a diverse range of features. A GANTT chart would be produced to allocate the sections of development and scale its feasibility. Furthermore as mentioned under the system design section, diagrams are to be created to understand the flow of the program.

Research and Planning

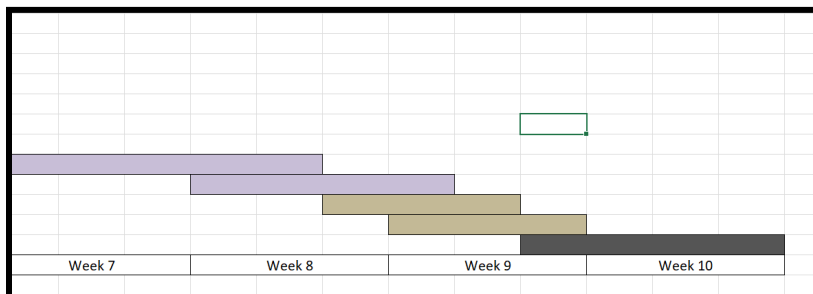
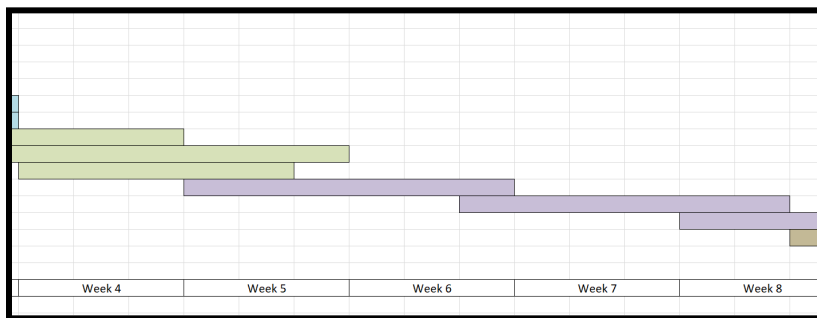
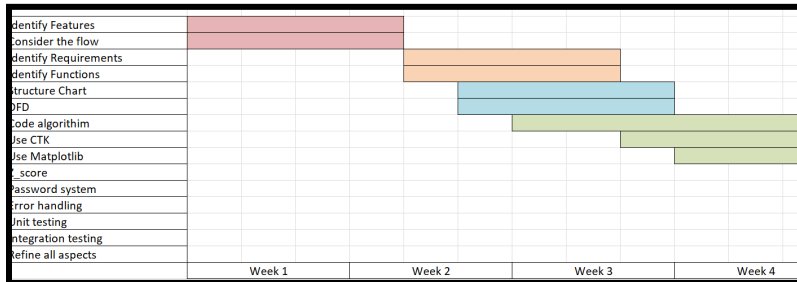
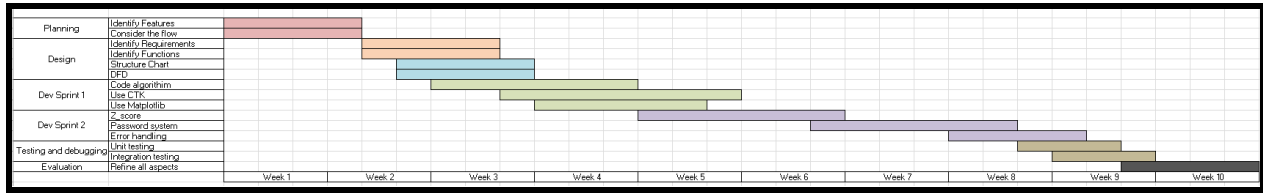
2.1 Development Methodology

The Agile method has been chosen to approach this project due to its flexibility and ability to scale down larger projects. The Agile methodology is a project management approach that concentrates on iterative development to adapt the project. This is done by breaking down work loads into multiple development sprints rather than focusing on a large uniform workload to fully complete the project. By breaking it down into parts, it allows change to be more flexible and increases scalability of the whole project by targeting and focusing on individual parts.

2.2 Tools and Technology

Python is the only coding language that is to be used in this subject. This is to increase ease of development as a result of extreme familiarity. Visual Studio is the IDE that should also be used due to its ability to be stored locally and also be connected to Github. Github is to be used to store files and documentation such as “read me”, creating a central location for all files to be stored. Regarding libraries, Custom Tkinter is the main one to be incorporated in order to provide an external UI alongside Matplotlib to graph results. Numpy is to be used for data calculations. Finally, Json and Hashlib are to be integrated for storing user data securely.

2.3 Gantt Chart and Planning Table



Note : Same Gaant chart, with the second to third screenshots just being the original one split into multiple screenshots

Stage	Start Week	End Week	Estimated Hours	Milestone
Planning	1	2	5	
	28-Apr	4-May	2	Identify Features
	28-Apr	4-May	3	Consider the flow of inputting marks and calculations
Design	2	3	10	
	7-May	14-May	2.5	Identify Requirements
	7-May	14-May	2.5	Determine the different functions present
	9-May	16-May	2.5	Draw a structure chart
	9-May	16-May	2.5	Draw diagram for flow of a user navigating the system and it's interaction
Development Sprint 1	3	5	15	
	12-May	23-May	7	Code algorithm to enter marks
	15-May	1-Jun	5	Learn and build UI using custom TK
	20-May	28-May	3	Use Matplot for graphing

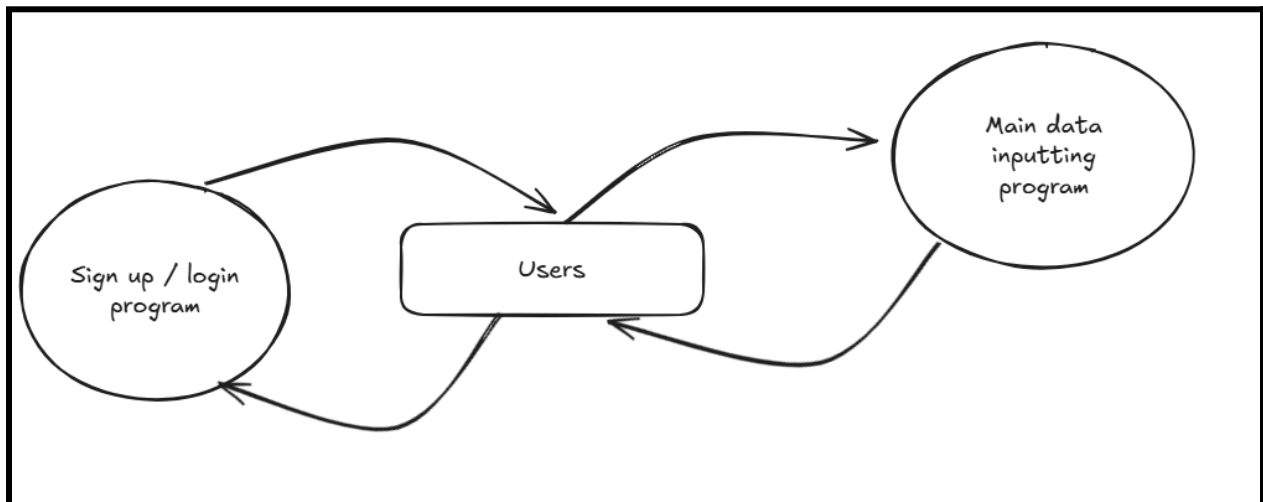
Development Sprint 2	5	7	15	
	25-May	6-Jun	7	Code Z score calculation rating and statistic calculation processes
	6-Jun	17-Jun	3	Password System
	17-Jun	22-Jun	5	Error handling
Testing & Debugging	7	9	10	
	20-Jun	24-Jun	3	Test the system as a whole
	22-Jun	28-Jun	3	Unit test each part
Evaluation & Deployment	9	10	3	
	28-Jun	3-Jul	3	Refine all aspects

2.4 Communication Plan

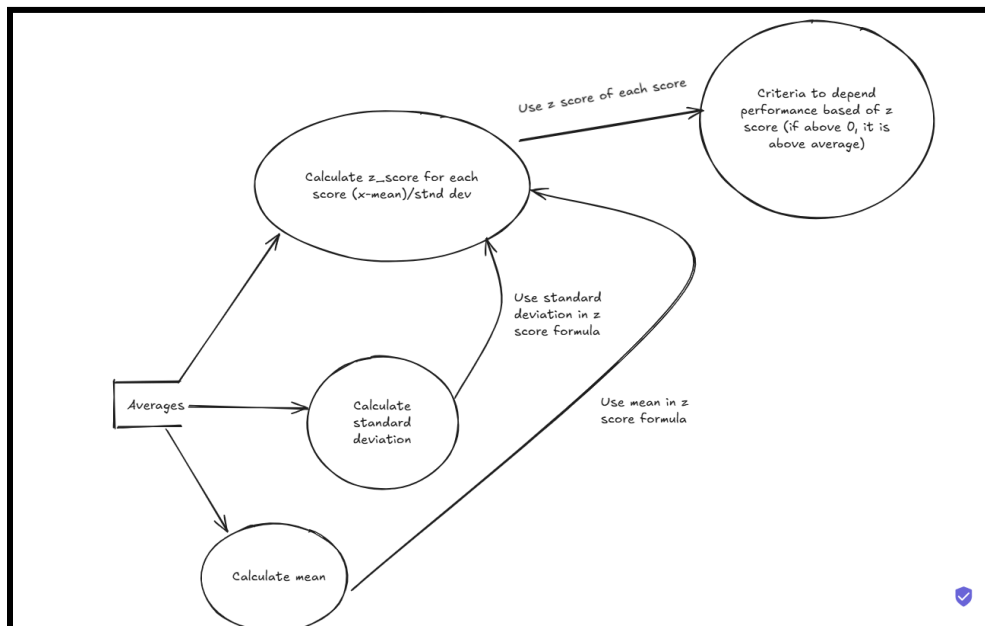
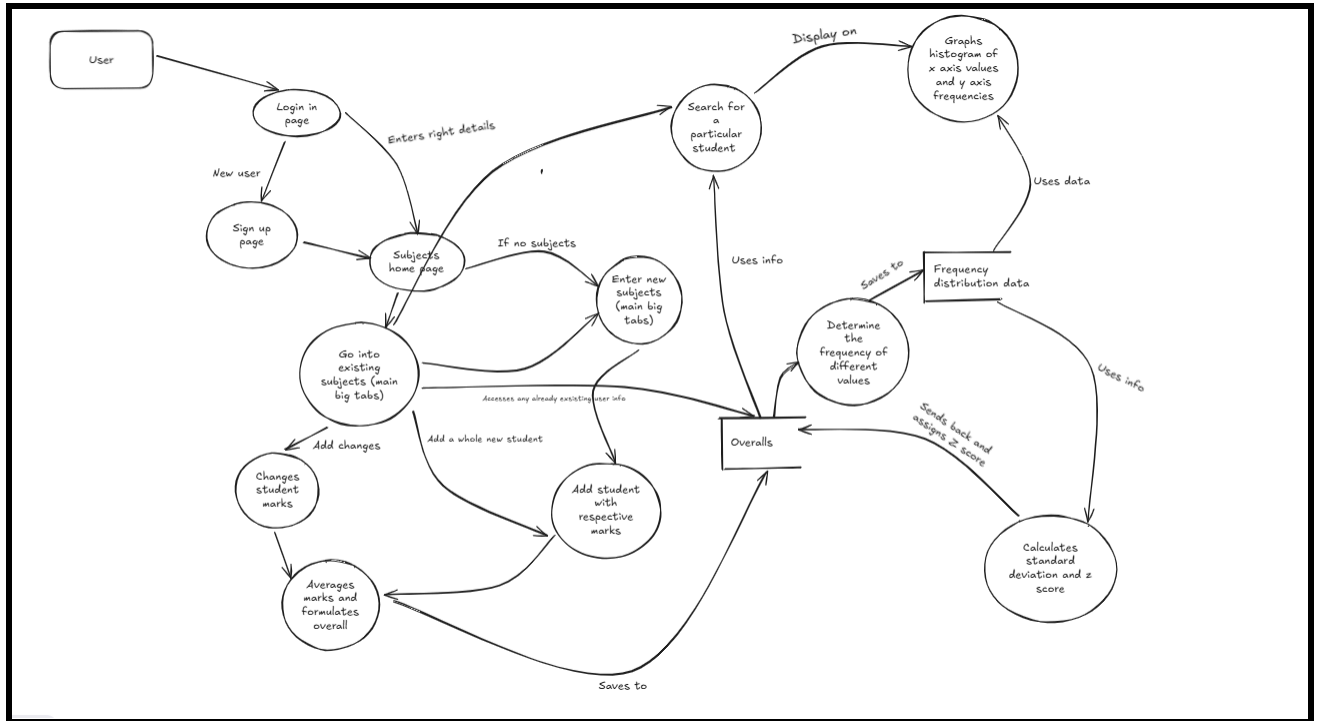
Feedback for the program would be gathered through user testing. User testing is a form of testing by having multiple users “trial” the program. By using this form of testing, feedback can also be gathered from their experiences and further enhance the program. This would also ensure the program is able to fulfil all the functional and nonfunctional requirements to a high standard.

System Design

3.1 Context Diagram

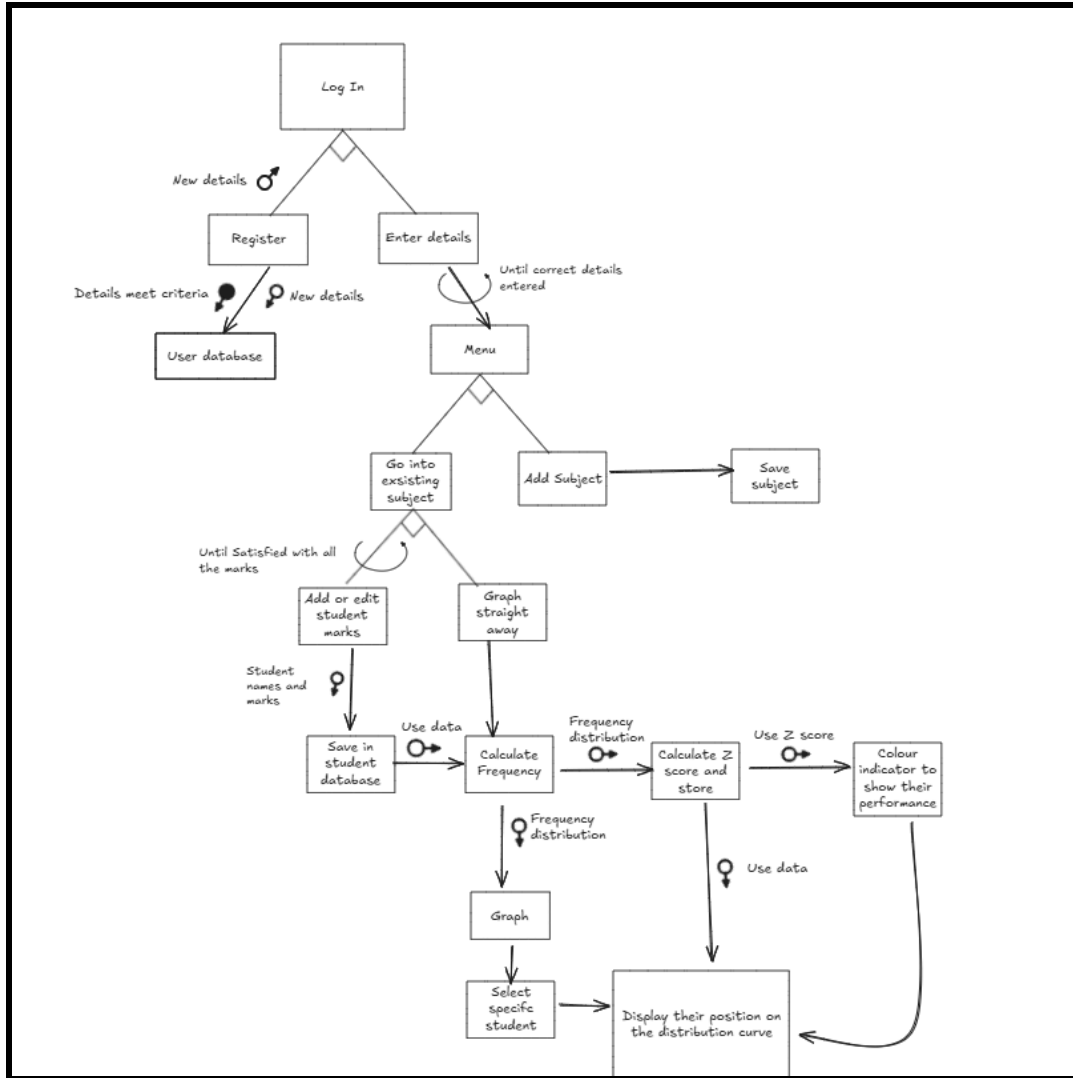


3.2 Data Flow Diagram (level 1 and 2 respectively)



Note : DFD level 2 focuses on the “calculates standard deviation and z score” process in the bottom right corner of DFD level 1

3.3 Structure Chart



3.4 IPO Chart

Input	Process	Output
Account credentials when signing up (username and password)	Saves them to a JSON file and hashes	Allows user to enter a program
Login details in order to login	Check them with the JSON file to see if the username matches the hashed password.	If the credentials match, it allows them to enter. Else, error message appears

Names of students	Saves them to a dictionary alongside there mark	Name appears in textbox and on diagram with individual marks
Marks of students	Saves them to two lists, one which has the raw values and one which averages them out the raw values by splitting them.	Marks appear on the diagram, with it being on y axis for the frequency histogram and individual marks column graph.
Marks of students	Saves them to a list, which splits it at commas and spaces. The numbers are then checked if they match a criteria and averaged.	Updates the list which is used for calculating performance
Averaged marks of students	Z score is calculated for each mark by first finding the standard deviation and mean. This is then plugged into the formula : $z\ score = \frac{score - average}{standard\ dev}$ The value of the z score is then used to judge student performance.	Shows student performance on scrollable frames in Custom Tkinter.
Inputted subjects	Creates an instance of the subject using the class. Additionally, it is saved to a dictionary with subjects	Subject is added to the subjects tab at the top.

3.5 Data Dictionary

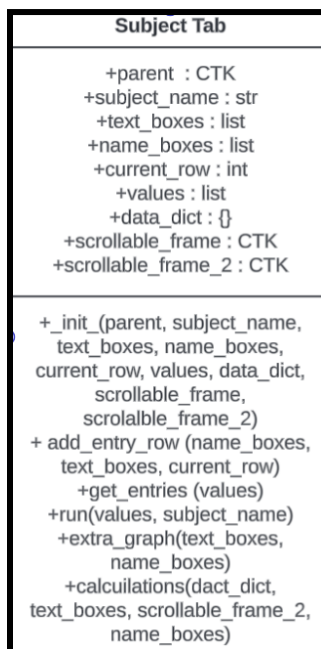
Name	Type	Desc
app	Widget	The main Customs Tkinter application variable
self.parent	Attribute	Acts as the parent window, with a new one created for

		each subject (parameter in class). It stores the two main scrollable frames for input and displaying z scores.
self.subject_name	Attribute	Attribute for subject names. It uses the name of the subject (parameter in class) entered in subject_name.
self.text_boxes	List	Adds marks for each student unique to each subject
self.name_boxes	List	Adds names for each student unique to each subject
self.values	List	Stores average marks if the user adds multiple marks to each text_boxes.
self.data_dict	Dictionary	Adds names of students and their respective Z scores
self.scrollable_frame	Widget	Adds a scrollable frame in each tab for the entry. It's master is the parent window for each tab
self.scrollable_frame_2	Widget	Adds a scrollable frame in each tab to display z_scores. It's master is the parent window for each tab
row_frame	Widget	New frame created every time a new entry is made.
avg	Int	Calculates average of the marks using the multiple entries
data	Array	Converts the marks list of averages to an array in order to be used for calculations
mean	Int	Finds mean of all the marks

		in data using numpy
names	List	List of all names to be assigned to z scores
marks	List	List of avg
z_scores	List	List of all z_scores. It is rounded to three decimal places
parts	List	List of the parts in each text box if multiple marks were typed in. All commas are first converted into spaces. Each string is split into its own list of parts
nums	Int	Converts string in parts to int
std_dev	Int	Finds the standard deviation for the set of data scores using numpy

3.6 Class Diagram

Since there is only one class, the following is the diagram for one of the classes :



Producing and Implementing

4.1 Development Process

In order to ensure the quality of the program and that it covers all functions and requirements, diagrams were first drawn in order to comprehend the scale of the program. For example, data flow diagrams were crucial in envisioning the cohesion of how a user interacted with the program, also providing insight into background processes. Additionally, fundamental features were first programmed to provide a “base” for other features to stem off from. For example, the login/sign up system was only integrated after the main program had been finalised, ensuring a foundation used to be built upon and thus aiding the development process. Furthermore, Agile is the primary design development approach to be used due to its ability to break down the project into smaller steps. The project has many features that have their own functions but also integrate with other features. In order to make this more scalable and approachable, the project was broken to development sprints with the process of each function being identified before attempting to understand how it would integrate amongst other functions.

4.2 Key Features Developed

The core features of this program are distinguished through the functions and classes of this program

Feature	Description and Justification
Subject Tab Class <pre>class SubjectTab: '''Class for the blueprint of each subject'''</pre>	This class provides a blueprint for all the functions of the program per subject tab, using the values entered in each tab for each subject's graphs and z scores. By having a class, it ensures values placed into each subject tab are unique to that tab and thus maintaining organisation.
Class initialization <pre>def __init__(self, parent, subject_name): # parent is the tab which widgets are put into and subject_name '''creates all the attributes for the class''' self.parent = parent self.subject_name = subject_name self.text_boxes = [] # marks for each student self.name_boxes = [] # names for each student self.values = [] # list to house the averages for each subjects self.data_dict = {} # dict for each person name to their z score self.scrollable_frame = ctk.CTkScrollableFrame(parent) # frame for adding rows for entering marks self.scrollable_frame.pack(pady=10, padx=10, fill="both", expand=True) self.scrollable_frame_2 = ctk.CTkScrollableFrame(parent) # frame for performance results aka z scores self.scrollable_frame_2.pack(pady=10, padx=10, fill="both", expand=True)</pre>	This defines all the attributes of the class and runs every time a new tab is created. It has the parameters of self (the copy of the tab), parent(houses all the widgets for each subject) and subject_name (name of the subject entered outside the class). It is essential to the class by defining all the attributes.
Get_entries	This function extracts all values entered in name_boxes and text_boxes and saves it to lists. Additionally, it splits the entries if

<pre>def get_entries(self): """Retrieves values from the entry boxes and prepares them for graphing""" self.values = [] # list for the averages (one as one in self) marks = [] # initial list for averages which is then placed into self.values if not self.name_boxes or not self.text_boxes: """If no name boxes or text boxes made (appears as false) and is flagged""" error_msg = ctk.CTkLabel(self.scrollable_frame, text="Please add at least one name and mark entry.", text_color="red", font=("calibri", 10)) error_msg.pack(pady=(10, 0)) return error_check = [entry.get() for entry in self.name_boxes + self.text_boxes] # creates a list containing both names and marks for entry in error_check: """for each name and mark it checks if the box is filled""" if not entry.strip(): """if by removing spaces at the end, there are no contents in the text box, it is flagged""" error_msg = ctk.CTkLabel(self.scrollable_frame, text="All fields must be filled out.", text_color="red", font=("calibri", 10)) error_msg.pack(pady=(10, 0)) return for entry in self.text_boxes:</pre>	<p>multiple were in the text box. This is crucial in bolstering the usability of the program, eliminating the need for users to formulate averages manually.</p>
<h3>Add_entry_row</h3> <pre>def add_entry_row(self): """function to add a new entry (name and mark) as a row""" row_frame = ctk.CTkFrame(self.scrollable_frame) # new frame for each row placed in the scrollable frame row_frame.pack(pady=10, padx=5, fill="X") nametext = ctk.CTkEntry(row_frame, width=200, placeholder_text="Name") # text box to add name nametext.pack(side="left", padx=(0, 10)) marktext = ctk.CTkEntry(row_frame, width=200, placeholder_text="Mark") # textbox to add mark marktext.pack(side="left", padx=(0, 10)) def delete_row(): """when x is pressed, this function runs""" row_frame.destroy() # deletes the frame for that specific row if nametext in self.name_boxes: self.name_boxes.remove(nametext) # removes appended names from the name box list if marktext in self.text_boxes: self.text_boxes.remove(marktext) # removes appended marks from the text box list</pre>	<p>This function is run every time button_add_row is pressed in order to create a new row to add a name and mark. This is done by creating a new frame for each row and placing widgets in this row.</p>
<h3>Run</h3> <pre>def run(self): """function for graphing""" plt.clf() counts, bins, patches = plt.hist(self.values, bins=10, color="skyblue", edgecolor="black", alpha=0.7) # counts is a list of frequency bin_centers = 0.5 * (bins[1:] + bins[:-1]) # finds center of each bin (by finding average from the edges of each bin) filtered_centers = [center for center, count in zip(bin_centers, counts) if count > 0] # has the centers of bins that aren't empty filtered_counts = [count for count in counts if count > 0] # contains the counts for non empty bins plt.plot(filtered_centers, filtered_counts, color="red", marker="o", linestyle="-", linewidth=2, label="Frequency Polygon") # plot plt.title("Histogram with Frequency Polygon ({self.subject_name})") plt.xlabel("value") plt.ylabel("frequency") plt.legend() plt.show()</pre>	<p>This function creates the graph using matplotlib. It first creates bins to group the data which allows a polygon to be graphed by connecting a red line to the centre of each bin. This function is essential to plotting the graph.</p>
<h3>Calculations</h3> <pre>def calculations(self): mean = np.mean(data) # calculates average of marks std_dev = np.std(data) # calculates standard deviation of marks z_scores = [round(x, 3) for x in ((data - mean) / std_dev)] if std_dev != 0 else [0 for _ in data] # calculates z scores for self.data_dict = {} # dict for name and z scores as established earlier for i in range(len(names)): # for the amount of names inputted self.data_dict[names[i]] = z_scores[i] # add that name and it's z score to the dictionary for key, value in self.data_dict.items(): # for each name and mark in the dictionary for x in marks: # creates a new frame for each student's performance item_frame = ctk.CTkFrame(self.scrollable_frame, 2) item_frame.pack(fill="X", pady=(2, 2)) label = ctk.CTkLabel(item_frame, text=f'{key}: {value} (avg)', font=("calibri", 10)) label.pack(side="left", padx=(0, 10)) # classifies a student's performance based on z scores. z scores that are more negative are more below the mean and z scores</pre>	<p>Calculates the z_scores of the scores. This is done by first extracting data from the name_boxes and text_boxes and putting them into lists. The list containing the extracted marks from text_boxes will then be converted into an array, before the mean for all scores was found using the array of marks. The standard deviation for each score was calculated before the z score for each score was found. Each z score was assigned to the person of that mark in the data dict. This function is necessary to determine the performance of each score.</p>
<h3>Add_subject</h3> <pre>def add_subject(): # function to add new subjects to a list as subject = subject_entry.get().strip() # gets the str of each subject if subject not in subject_tabs: # adds to list if doesn't already exist subject_tabs.add(subject) # adds to the list of the tabs subject_tabs[subject] = subject_tabs(tabview.tab(subject), subject) # creates a new instance of the class with the subject subject_entry.delete(0, "end") # empties the entry box after adding the subject add_subject_button = ctk.CTkButton(subject_entry_frame, text="Add Subject", command=add_subject) add_subject_button.pack(side="left")</pre>	<p>This function adds inputted subjects to the tab_view on the top of the ctk window. It first checks if there is any input in the subject_entry text box and whether the subject already exists in the subject_tabs dictionary. By doing so, it allows the input of subjects to be handled correctly.</p>
<h3>Master_program</h3>	<p>This is the function that stores the entire</p>

```
def master_program():
    """main function that houses everything"""
    clear_window() # clears signup and login

    title_label = tkinter.Label(app, text="Student Dashboard Manager", font=("Calibri", 20, "bold"))
    title_label.pack(pady=(20, 10))

    # Subject input
    subject_entry_frame = tkinter.Frame(app) # creates a frame at the top to input subjects
    subject_entry_frame.pack(pady=(10, 0))

    subject_entry = tkinter.Entry(subject_entry_frame, placeholder="Enter new subject")
    subject_entry.pack(side="left", padx=(0, 10))

    tabview = tkinter.Tabview(app) # creates tabs at the top of the screen
    tabview.pack(side="top", padx=10, fill="both", expand=True)
```

main program, with the function running after the login/signup criteria have been satisfied. This enables the smooth transition from the sign up/login system to the main program.

Login

```
def login():
    """first page that appears that allows users to login to the system"""
    clear_window()

    username = tkinter.Entry(app, placeholder="Enter username", width=200, height=40, font=("Arial", 16))
    username.pack(pady=(20, 10))
    password = tkinter.Entry(app, placeholder="Enter password", width=200, height=40, font=("Arial", 16), show="*")
    password.pack(pady=(10, 20))

    def confirm_login():
        """Checks if the inputted username and password are valid"""
        login_username = username.get() # retrieves username and password from textboxes
        login_password = password.get()
        if not login_username or not login_password:
            """If the values from the text boxes come back as false"""
            label = tkinter.Label(app, text="Username and password cannot be empty.", font=("Arial", 16), text_color="red") # flagged as being empty
            label.pack(pady=(10, 20))
            return
        with open("user.json", "r") as f:
            """opens json file to use for validation"""
            data = json.load(f)
        if login_username in data and data[login_username] == hash_password(login_password):
            """Checks if the username exists in the database and if the inputted username and password match with the username and hashed password"""
            master_program() # runs main program
        else:
            label = tkinter.Label(app, text="Invalid username or password. Please try again.", font=("Arial", 16), text_color="red") # error message
            label.pack(pady=(10, 20))

    confirm = tkinter.Button(app, text="Confirm", width=200, height=40, font=("Arial", 16), command=confirm_login)
    confirm.pack(pady=(10, 20))

    signup = tkinter.Button(app, text="Sign Up", width=200, height=40, font=("Arial", 16), command=signup)
    signup.pack(pady=(10, 20))
```

The function is first run before signup, allowing users to input details to login, making sure the inputted username and password match saved data in the Json file. This is crucial in meeting the requirements of producing secure software architecture, complying with industry standards by also using hashing in the hash function.

Sign_up

```
def hash_password(password): # hashes the password
    """takes in the password and hashes it"""
    return hashlib.sha256(password.encode()).hexdigest()

def sign_up(): # sign up function
    clear_window()

    """text boxes for username and password"""
    username = tkinter.Entry(app, placeholder="Enter username", width=200, height=40, font=("Arial", 16))
    username.pack(pady=(20, 10))
    password = tkinter.Entry(app, placeholder="Enter password", width=200, height=40, font=("Arial", 16), show="*")
    password.pack(pady=(10, 20))

    def confirm_signup(): # checks if the username and password are valid
        """gets the username and password entered"""
        saved_username = username.get()
        saved_password = password.get()
        if not saved_username or not saved_password:
            """If both username and password are false/don't exist"""
            label = tkinter.Label(app, text="Username and password cannot be empty.", font=("Arial", 16), text_color="red") # error message
            label.pack(pady=(10, 20))
            return
        with open("user.json", "r") as f: # loads json file as a python dictionary
            data = json.load(f)
        if saved_username in data:
            """Checks if the username already exists in the json file"""
            label = tkinter.Label(app, text="Username already exists. Please try again.", font=("Arial", 16), text_color="red")
            label.pack(pady=(10, 20))
        else:
            """passes criteria and saves to json"""
            data[saved_username] = hash_password(saved_password) # puts the username and the hashed version of password
            with open("user.json", "w") as f: # opens json and adds data to it
                json.dump(data, f, indent=4)
            label = tkinter.Label(app, text="Sign up successful! Please log in.", font=("Arial", 16), text_color="green") # confirmation message
            label.pack(pady=(10, 20))

    confirm = tkinter.Button(app, text="Confirm", width=200, height=40, font=("Arial", 16), command=confirm_signup)
    confirm.pack(pady=(10, 20))

    back = tkinter.Button(app, text="Back to login", width=200, height=40, font=("Arial", 16), command=login)
    back.pack(pady=(10, 20))
```

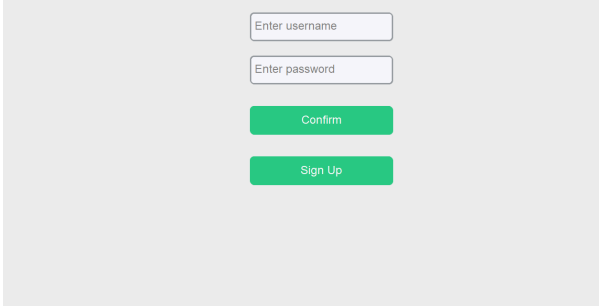
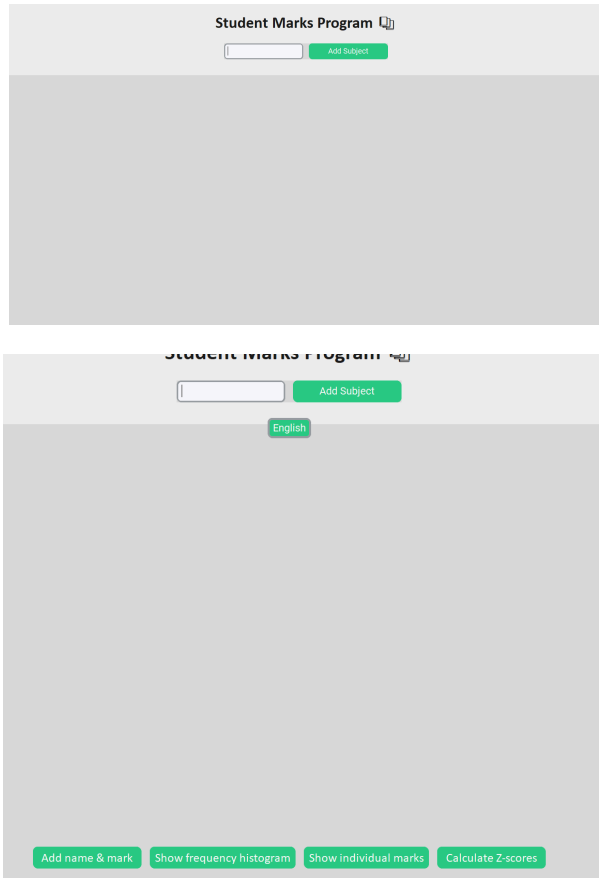
Similarly to login, it allows users to create an account, ensuring it meets the criteria of having a unique username. This is crucial in meeting the requirements of functionality in establishing the basis of secure software architecture.

Extra_graph

```
def extra_graph(self):
    names = [entry.get() for entry in self.name_boxes] # creates a list for names
    marks = [] # list for marks for the bar graph (separate from frequency histogram)
    ## SOME STRIPPING AND AVERAGING FORMULA AS BEFORE ##
    for entry in self.text_boxes:
        raw = entry.get()
        parts = [x for x in raw.replace(' ', ' ').split() if x.strip()]
        nums = [int(x) for x in parts]
        if nums:
            avg = sum(nums) / len(nums)
            marks.append(avg)
        else:
            marks.append(0)
    plt.clf() # clears any previous graphs in the matplotlib lib
    plt.bar(names, marks, color="skyblue", edgecolor="black")
    plt.title(f"Average Marks per Student ({self.subject_name})")
    plt.xlabel("Student")
    plt.ylabel("Average Mark")
    plt.show()
```

This plots an extra graph, using names list as the x axis and marks list as the y axis. This is essential in providing variety for the user for different ways to graphically show student marks, increasing understanding.

4.3 Interface

Screen Shots	Desc
	<p>This is the login page of the system which is the first window that appears when the program is run. It allows users that have already signed up to input their username and password. If a user has yet to sign up, they can press the sign up button to go to redirect them to sign up.</p>
	<p>This is the main page once a user has logged in. They can type in the name of a subject and press add subject to add it to their tabs. For example, if a user was to write “English”, “English” would appear at the top of the tabview.</p>

English Engineering

Mark	90	X
Alex	80	X

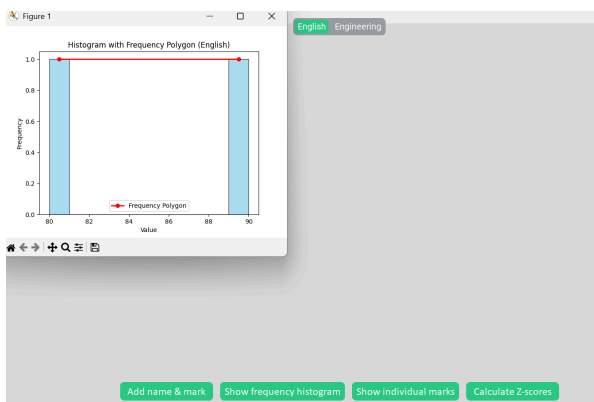
Add name & mark Show frequency histogram Show individual marks Calculate Z-scores

The user can add the name and mark of a student by pressing “Add Name and Mark”. Additionally, marks are unique for each subject. Furthermore, by pressing the “X”, it deletes the frame and therefore the entry of that row.

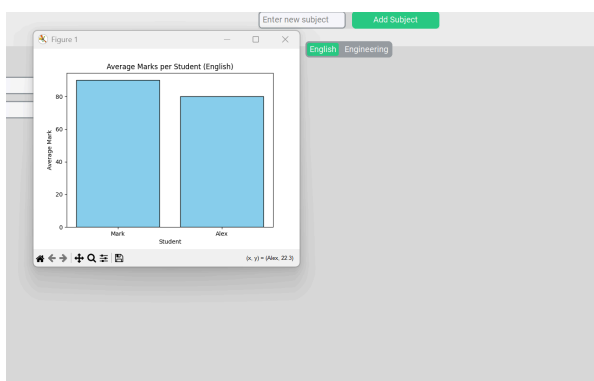
Enter new subject Add Subject

English Engineering

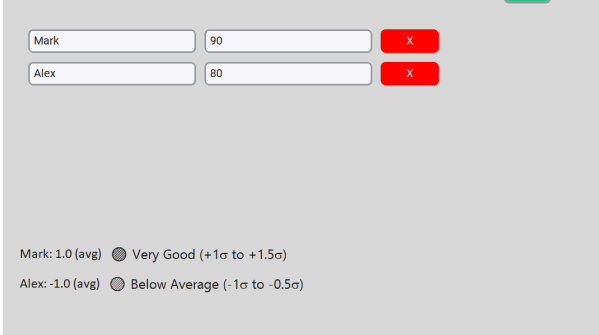
Mark 40 X



Once marks are entered, users can press “add frequency histogram” to see a graph of a histogram, with frequency graphed on the y axis and the types of scores graphed on the x axis. Additionally, a polygon is added for interpolation.



Users can also show a bar graph of the student’s individual marks by pressing “Show Individual Marks” compared against each other.

	<p>An overall summary of each student is also given when “Show performance” is pressed. It shows each student’s z_score and also their performance compared to the average. The text in brackets show the range of standard deviations that they lie from the mean.</p>
---	---

Testing and Evaluating

5.1 Testing Methods Used

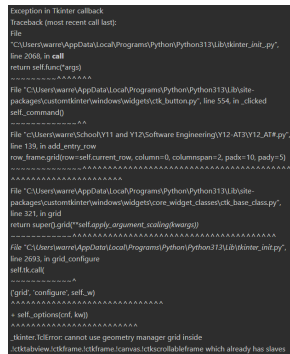
Testing is a process in Agile development, which ensures that the system functions properly, meeting criteria derived in previous stages of development. In order to effectively debug the program, there are multiple methods of testing as defined below :

- **Unit Testing** : This is done by breaking the main program into individual functions and modules. Each individual component is then tested to identify and address separate bugs.
- **Integration Testing** : This is opposite to unit testing, where individual components are combined. This provides understanding of how different features interact with one another
- **Acceptance Testing** : Runs the program with all the components to see how it operates in a real world scenario.
- **User Testing** : Allows users to run the program and also observe possible errors that may occur in real world usage.

5.2 Testing Logs

Name	Type	Desc	Results
Opening Json file and inputting usernames and passwords	Unit testing	Unit testing was used to first determine the fundamentals of opening json files before integrating	Fail. It was determined that each user had to have a key with their name and a value of their

		with the whole program.	password.
Opening Json file and extracting usernames and passwords	Unit testing	Unit testing was applied to see if the program was able to extract information from the json file	Pass
Creating the sign up page to input usernames	Integration Testing	The sign up function as a whole integrated the input features	Pass
Creating the login page	Integration testing	Integration testing determined if the login page was able to work alongside the sign up function, using newly inputted usernames and passwords.	Pass
Determining if passwords or usernames exists	Unit testing	Unit testing was done to see if a for loop was successful in reading a translated json file.	Pass
Integrating sign up/login with main program	Integration Testing	Integration testing was applied to see if containing the main program in a function would work	Pass
Classes for each subject	Acceptance and integration testing	Acceptance testing was done by running the class as a whole by giving an example instance of a subject. Integration testing was done to observe how	Pass
Histograms for subjects	Unit testing	Unit testing was done by creating a	Fail. The values plotted on the x axis

		separate file to run and create a distribution.	had to be filtered from scores with 0 frequency.
Calculating z scores	Unit testing	The process of gathering data from lists and calculating the z scores. Unit testing was done to ensure that this function worked before sorting it out in the main program	Fail. the entry.get() was needed in order to convert the widget objects to the values in the widget boxes. Without entry.get(), it saved values entered in the textbox as widgets rather than the values put in the text box.
Error handling for textboxes with no content	Integration testing	The ability for the program to place an error message while disrupting the rows was done through integration testing and was tested alongside the function of adding rows.	Fail. Grid and pack cannot be mixed in the same parent widget. 
Determining nature of values in widgets	Integration Testing	In order to add further error handling to ensure all marks are values, testing was done to determine the original nature (string or int) of marks when extracted from input.	Pass
Ability to average marks	User testing and unit testing	The ability for a string to be broken down was tested separately	Pass

		from the program. This was given to users to have feedback on it's operation (e.g if when users inputted marks, it correctly averages their marks)	
--	--	---	--

5.3 Evaluation Against Requirements

The solution produced has covered all the components required in the criteria, meeting the goals established at the beginning of the Agile process. For example, functional requirements have all been met such as having the foundations of a sign up login system and allowing users to input their own subjects. Additionally, some non - functional requirements have been met, including an easy to navigate UI and a hashing process to increase security. By meeting these requirements, it ensures that users have access to all the basic features in the program.

5.4 Improvements and Future Work

Although the program has met all functional requirements, some of the non - functional requirements were met to their full potential. The ability for the program to display where a student is on the polygon could have been integrated, making the program more visually appealing and providing features that may help users better analyse student performance compared to the cohort. Furthermore, the ability for subjects to be saved for each student after the runtime session concluded was planned to be added yet wasn't due to time constraints.

Feedback and Reflection

6.1 Client Feedback

Client feedback was collected at the end of development and throughout several stages. The following were common points of feedback :

Name	Feedback
A	<ul style="list-style-type: none"> Enable the ability for users to write multiple marks in one textbox for each student, representing percentages they achieved in the assessment tasks. Average these marks to find the

	overall percentage, eliminating the need for teachers to manually figure it out.
B	<ul style="list-style-type: none"> Allow the program to save subjects for each user even after the runtime session is over.
C	<ul style="list-style-type: none"> The interface should be further modernised, specifically coloured emojis for the z_score system
D	<ul style="list-style-type: none"> It should give a deeper summary of each student, with maybe a tab for each student in a new window

6.2 Personal Evaluation

Many valuable skills were learnt and developed through this project. For example, this project allowed me to experiment with external UI's rather than being confined to the terminal, understanding how spacing features such as packing works. Furthermore, the basics of user security were developed through the username password system, understanding how to save information in a json file. Additionally, software design skills were further refined through working on diagrams.

Appendix

Log

#	Date	Desc
1	May 19th	<p>Created the first tk window with the title as well and two tabs.</p> <pre> app = tk.Tk() app.geometry("700x1000") app.title("Program") tabview = tk.TkTabview(app) tabview.pack(fill="both", expand=True, padx=5, pady=5) tabview.add("Data Entry") tabview.add("Table") entry_tab = tabview.tab("Data Entry") title_label = tk.Label(entry_tab, text="Student Marks Program", font=("Calibri", 20, "bold")) title_label.pack(pady=(20, 10)) </pre>

2	May 22nd	<p>Through the design stage, a scrollable frame was identified to be the most suitable option as the main parent frame, allowing users to scroll.</p> <pre>scrollable_frame = ctk.CTkScrollableFrame(entry_tab) scrollable_frame.pack(pady=20, padx=20, fill="both", expand=True)</pre> <p>Additionally, buttons were added to the main frame.</p> <pre>title_label.pack(pady=(20, 10)) button_add_row = ctk.CTkButton(entry_tab, text="Add name & mark", font=("Calibri", 16)) button_add_row.pack(pady=10) button_run = ctk.CTkButton(entry_tab, text="Show frequency histogram", font=("Calibri", 16)) button_run.pack(pady=10) extra_button = ctk.CTkButton(entry_tab, text="Show individual marks", font=("Calibri", 16)) extra_button.pack(pady=10)</pre>
3	May 23rd	<p>Text boxes for names and the mark values were added to the program within the master scrollable frame. Lists were also made for names and marks to append the user input.</p> <pre>name_boxes.append(nametext) # Adds the name to a list to save for future purpose text_boxes.append(newtext) # Adds the mark to a list to save for future purposes</pre> <pre>nametext = ctk.CTkEntry(scrollable_frame, width=200) # placed in the row frame and is the input for the name nametext.pack(side="left", padx=(0, 10)) newtext = ctk.CTkEntry(scrollable_frame, width=200) # placed in the row frame and is the input for the mark newtext.pack(side="left", padx=(0, 10))</pre>
4	May 23rd	<p>Changes were made to the rows, with each row now receiving their own frame in order to create a delete feature later on.</p> <pre>def add_entry_row(): # Adds a row for name and marks simultaneously on the left side of the page global current_row row_frame = ctk.CTkFrame(scrollable_frame) # Each row has it's own frame which is placed in the master scrollable frame row_frame.grid(row=current_row, column=0, columnspan=2, padx=10, pady=5, sticky="nw") #the frame is placed in whatever current r nametext = ctk.CTkEntry(row_frame, width=200) # placed in the row frame and is the input for the name nametext.pack(side="left", padx=(0, 10)) newtext = ctk.CTkEntry(row_frame, width=200) # placed in the row frame and is the input for the mark newtext.pack(side="left", padx=(0, 10))</pre>
5	May 25th	<p>Added delete row feature.</p> <pre>def delete_row(): row_frame.destroy() # Deletes the frame of the row # Remove the entries from the lists if nametext in name_boxes: name_boxes.remove(nametext) if newtext in text_boxes: text_boxes.remove(newtext) del_button = ctk.CTkButton(row_frame, text="X", command=delete_row, fg_color="red", width=70) # Button to delete del_button.pack(side="left")</pre>
6	May 28th	<p>get_entries() was made to extract the marks in the list and convert them to be integers.</p> <pre>def get_entries(): global values raw_values = [entry.get() for entry in text_boxes] # Retrieve values in the list for marks values = [int(x) for x in raw_values] # Makes them all integers to be used in the graph run()</pre>
7	May 28th	<p>run() was made to create the matplotlib histogram using the values list created earlier.</p>

		<pre> def run() : plt.clf() # Clears any current graph global values counts, bins, patches = plt.hist(values, bins=10, color="skyblue", edgecolor="black", alpha=0.7) # Calculate bin centers plt.xlabel("Value") plt.ylabel("Frequency") plt.legend() plt.show() </pre>
8	May 29th	<p>Changes were made to exclude bins with zero frequencies to prevent them from being plotted</p> <pre> filtered_centers = [center for center, count in zip(bin_centers, counts) if count > 0] filtered_counts = [count for count in counts if count > 0] </pre>
9	May 29th	<p>Frequency polygon was added, with the bin centers first being identified before a line graph was integrated.</p> <pre> bin_centers = 0.5 * (bins[1] + bins[-1]) # calculate bin centers for the polygon # Exclude bins with 0 frequency filtered_centers = [center for center, count in zip(bin_centers, counts) if count > 0] filtered_counts = [count for count in counts if count > 0] # Plot frequency polygon only for non-zero frequencies plt.plot(filtered_centers, filtered_counts, color="red", marker="o", linestyle="--", linewidth=2, label="Frequency Polygon") plt.title("Histogram with Frequency Polygon") plt.xlabel("Value") plt.ylabel("Frequency") </pre>
10	May 30th	<p>An option to create a column graph was added.</p> <pre> extra_button = ctk.CtkButton(entry_tab, text = "Show individual marks", command = extra_graph, font=("Calibri", 10)) extra_button.pack(pady=10) def extra_graph() : global values global text_boxes plt.clf() plt.bar(name_boxes, text_boxes, color="skyblue", edgecolor="black") # Create bar chart plt.show() </pre>
10	May 30th	<p>An error was encountered regarding the format of name_boxes and text_boxes in the new graph. Changes were made</p> <pre> def extra_graph() : global values global text_boxes plt.clf() names = [entry.get() for entry in name_boxes] # Extract names from name_boxes marks = [int(entry.get()) for entry in text_boxes] # Extract marks from text_boxes plt.bar(names, marks, color="skyblue", edgecolor="black") # Create bar chart plt.show() </pre>
11	June 1st	<p>Calculation system is started to be worked on</p> <pre> def calculations() : data = np.array(values) mean = np.mean(data) std_dev = np.std(data) z_scores = [(data - mean)/ std_dev] </pre>
12	June 3rd	<p>Changes were made to the z scores list, by using the list function instead. However,</p>

		<p>when put into a dictionary with names as keys and z scores as values, the program doesn't print anything.</p> <pre>def calculations() : global names global marks names = [entry.get() for entry in name_boxes] # Extract names from name_boxes marks = [int(entry.get()) for entry in text_boxes] # Extract marks from text_boxes data = np.array(marks) mean = np.mean(data) std_dev = np.std(data) z_scores = list((data - mean)/ std_dev) formatted_z_scores = [f"{z:.2f}" for z in z_scores] for i in range(len(names)): dict[names[i]] = formatted_z_scores[i] dict_label.configure(text=str(dict)) tableView.set("Table")</pre>
13	June 9th	<p>On the new data tab, a row frame similar to the one in the entry tab was added. This was to display the z scores as a label on each row, with each row being its own frame.</p> <pre>for key, value in dict.items(): label = ctk.CTkLabel(row_frame_2, text = key + ":" + str(value), font = ("Calibri", 16)) # label.pack(pady=(20, 10)) # Pack the label into the scrollable frame current_row_2 += 1 # Increment the row counter for the next label</pre>
14	June 11th	<p>In order to classify those with higher and lower z scores, an if else statement was used depending if they were below or above 0.</p> <pre>if value > 0 : sum_label = ctk.CTkLabel(item_frame, text = "Good", font = ("Calibri", 16), width = 70) sum_label.pack(side = "left") elif value == 0 : sum_label = ctk.CTkLabel(item_frame, text = "Average", font = ("Calibri", 16), width = 70) sum_label.pack(side = "left") else : sum_label = ctk.CTkLabel(item_frame, text = "Poor", font = ("Calibri", 16), width = 70) sum_label.pack(side = "left")</pre>
15	June 18th	<p>Further detail was added to the z score classification system, making it easier to differentiate marks.</p> <pre>if value > 0 and value < 1: sum_label = ctk.CTkLabel(item_frame, text="👍 Good", font=("Segoe UI Emoji", 16), width=100) elif value == 1 and value < 2: sum_label = ctk.CTkLabel(item_frame, text="👏 Excellent", font=("Segoe UI Emoji", 16), width=100) elif value >= 2 and value < 3: sum_label = ctk.CTkLabel(item_frame, text="👑 Outstanding", font=("Segoe UI Emoji", 16), width=100) elif value < 0 and value >= -1: sum_label = ctk.CTkLabel(item_frame, text="👎 Below Average", font=("Segoe UI Emoji", 16), width=100) elif value <= -1 and value >= -2: sum_label = ctk.CTkLabel(item_frame, text="👎 Average", font=("Segoe UI Emoji", 16), width=100) elif value < -2 and value >= -3: sum_label = ctk.CTkLabel(item_frame, text="👎 Needs Improvement", font=("Segoe UI Emoji", 16), width=100) else: sum_label = ctk.CTkLabel(item_frame, text="👎 Poor", font=("Segoe UI Emoji", 16), width=100) sum_label.pack(side="left")</pre>
16	June 18th	<p>The two tabs were removed as it added confusion to where different features were located. Instead, all features are located on the default parent tab.</p>

		
17	June 21st	<p>The feature to add multiple subjects and have designated marks for each subject was created. This was done by incorporating object oriented programming. Everytime a new subject is created, it uses the class as a blueprint.</p> <pre>def add_subject(): subject = subject_entry.get().strip() if subject and subject not in subject_tabs: tabview.add(subject) subject_tabs[subject] = SubjectTab(tabview.tab(subject), subject) subject_entry.delete(0, "end") add_subject_button = ctk.CtkButton(subject_entry_frame, text="Add Subject", command=add_subject) add_subject_button.pack(side="left")</pre> <pre>class SubjectTab: def __init__(self, parent, subject_name): self.parent = parent self.subject_name = subject_name self.text_boxes = [] self.name_boxes = [] self.current_row = 0 self.values = [] self.data_dict = {} # frame for entry rows self.scrollable_frame = ctk.CtkScrollableFrame(parent) self.scrollable_frame.pack(pady=10, padx=10, fill="both", expand=True) # frame for results self.scrollable_frame_2 = ctk.CtkScrollableFrame(parent) self.scrollable_frame_2.pack(pady=10, padx=10, fill="both", expand=True)</pre>
18	June 25th	<p>A login feature was added. It was first unit tested before being integrated with the main program. The main change was the order being shifted, with a lot of the main master program initialization put at the start and a large function was made to encapsulate the main program.</p> <pre>import tkinterinter as ctk import numpy as np import matplotlib.pyplot as plt import json import os ctk.set_appearance_mode("light") ctk.set_default_color_theme("green") # Ensure data.json exists if not os.path.exists("data.json"): with open("data.json", "w") as f: json.dump({}, f) app = ctk.Ctk() app.geometry("1100x900") app.title("Student Marks Program") def close_window(): for widget in app.winfo_children(): widget.destroy() def sign_up(): clear_window() username = ctk.CtkEntry(app, placeholder_text="Enter username", width=200, height=40, font=("Arial", 16)) username.pack(pady=(20, 10)) password = ctk.CtkEntry(app, placeholder_text="Enter password", width=200, height=40, font=("Arial", 16), show="*") password.pack(pady=(10, 30)) def confirm_signup(): saved_username = username.get() saved_password = password.get() with open("data.json", "r") as f:</pre>

		<pre> if saved_username in data: label = ctk.CTkLabel(app, text="Username already exists. Please try again.", font=("Arial", 16), text_color="red") label.pack(pady=(10, 20)) else: data[saved_username] = saved_password with open("data.json", "w") as f: json.dump(data, f, indent=4) label = ctk.CTkLabel(app, text="Sign up successful! Please log in.", font=("Arial", 16), text_color="green") label.pack(pady=(10, 20)) confirm = ctk.CTkButton(app, text="Confirm", width=200, height=40, font=("Arial", 16), command=confirm_signup) confirm.pack(pady=(10, 20)) back = ctk.CTkButton(app, text="Back to login", width=200, height=40, font=("Arial", 16), command=login) back.pack(pady=(10, 20)) def login(): clear_window() username = ctk.CTkEntry(app, placeholder_text="Enter username", width=200, height=40, font=("Arial", 16)) username.pack(pady=(20, 10)) password = ctk.CTkEntry(app, placeholder_text="Enter password", width=200, height=40, font=("Arial", 16), show="*") password.pack(pady=(10, 20)) def confirm_login(): login_username = username.get() login_password = password.get() with open("data.json", "r") as f: data = json.load(f) if login_username in data and data[login_username] == login_password: launch_marks_program() else: label = ctk.CTkLabel(app, text="Invalid username or password. Please try again.", font=("Arial", 16), text_color="red") label.pack(pady=(10, 20)) confirm = ctk.CTkButton(app, text="Confirm", width=200, height=40, font=("Arial", 16), command=confirm_login) confirm.pack(pady=(10, 20)) signup = ctk.CTkButton(app, text="Sign up", width=200, height=40, font=("Arial", 16), command=signup) signup.pack(pady=(10, 20)) def launch_marks_program(): clear_window() </pre>
19	June 30th	<p>Error handling was first added to the system. The first part of error handling was in the username and password system by ensuring the textboxes had to be filled (for an error could occur if someone signed up with no username or password which was save as " " : " " in the json file.</p> <pre> def confirm_login(): login_username = username.get() login_password = password.get() if not login_username or not login_password: label = ctk.CTkLabel(app, text="Username and password cannot be empty.", font=("Arial", 16), text_color="red") label.pack(pady=(10, 20)) return with open("user.json", "r") as f: data = json.load(f) </pre>
20	July 2nd	<p>Error handling was added to the master program, primarily through the use of if not, which checks if any criteria (e.g if there are marks entered or not) is false, which therefore is flagged. This was added to ensure all names and marks were filled out and that the marks were numbers.</p> <pre> if not self.name_boxes or not self.test_boxes: error_msg = ctk.CTkLabel(self.scrollable_frame, text="Please add at least one name and mark entry.", text_color="red", font=("Calibri", 16)) error_msg.pack(pady=(10, 0)) return error_check = [entry.get() for entry in self.name_boxes + self.test_boxes] for entry in error_check: if not entry.strip(): error_msg = ctk.CTkLabel(self.scrollable_frame, text="All fields must be filled out.", text_color="red", font=("Calibri", 16)) error_msg.pack(pady=(10, 0)) return for entry in self.test_boxes: raw = entry.get() parts = [x for x in raw.replace(',', ' ').split()] num = [] for x in parts: try: num.append(int(x)) except ValueError: error_msg = ctk.CTkLabel(self.scrollable_frame, text="All marks must be numbers.", text_color="red", font=("Calibri", 16)) error_msg.pack(pady=(10, 0)) return </pre>

21	July 3rd	<p>The performance classification was slightly changed to cover a greater range of standard deviations to further differentiate students.</p> <pre> label.pack(side="left", padx=(0,10)) if value >= 7: sum_label = tk.Label((item_frame, text=f"🏆 Top Performer (+2SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value == 1.5: sum_label = tk.Label((item_frame, text=f"🌟 Excellent (+1.5SD to +2SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= 1: sum_label = tk.Label((item_frame, text=f"🌟 Very Good (+SD to +1.5SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= 0.5: sum_label = tk.Label((item_frame, text=f"🌟 Above Average (+0.5SD to SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= 0: sum_label = tk.Label((item_frame, text=f"🌟 Average (0 to +0.5SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= -0.5: sum_label = tk.Label((item_frame, text=f"🌟 Slightly Below Average (-0.5SD to 0) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= -1: sum_label = tk.Label((item_frame, text=f"🌟 Below Average (-SD to -0.5SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= -1.5: sum_label = tk.Label((item_frame, text=f"🌟 Needs Improvement (-1.5SD to -SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) elif value >= -2: sum_label = tk.Label((item_frame, text=f"🌟 Poor (-2SD to -1.5SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) else: sum_label = tk.Label((item_frame, text=f"🌟 Very Poor (-2.5SD to -2SD) with score {x}", font=("Segoe UI Emoji", 16), width=150) </pre>
22	July 3rd	<p>Hashing was added to ensure further security. This was first done by importing hashlib. The hash password takes in the password as plain text and returns as a hashed version. When a user tries to login, the password they entered is hashed and checked if it matches the hashed password in the json file.</p> <pre> def hash_password(password):# hashes the password '''takes in the password and hashes it''' return hashlib.sha256(password.encode()).hexdigest() # saves criteria and saves to json data[saved_username] = hash_password(saved_password) # puts the username and the hashed version of password data = json.load(f) if login_username in data and data[login_username] == hash_password(login_password): '''checks if the username exists in the database and if the inputted username and password match''' master_program() #runs main program else: </pre>

GitHub Link

<https://github.com/Warrenmao1/Y12-AT3>