# Criterion B

1. **Decomposition Diagram (top-down)**: Constructing the flow of operation between various functions to map out the functionality of the system. This brainstorm will ease development in a later stage.

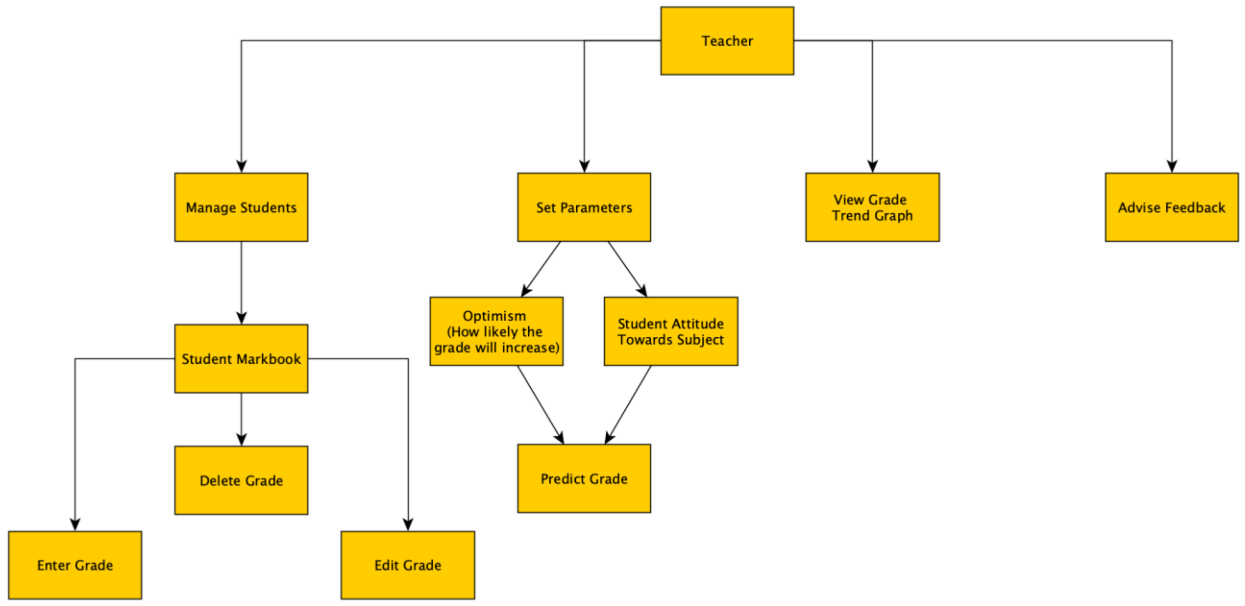


Figure 2: Top Down Diagram

2. **UML Class Diagrams:** Shows a first-brainstorm of the class diagrams I predict I will need my coding. I expect to develop them as the IA process continues.

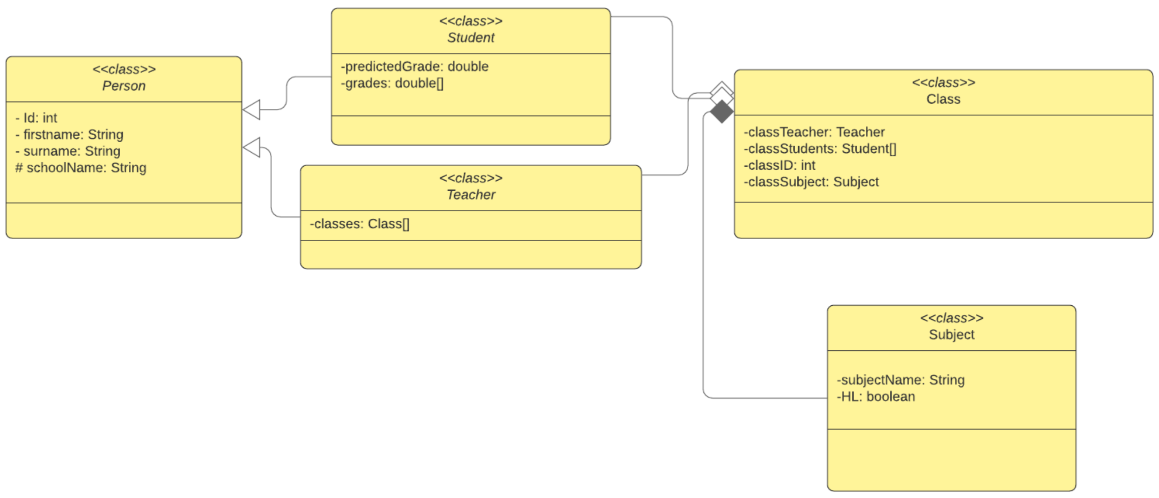


Figure 3: Initial UML Diagram

The next diagram shows a more developed plan of the system with a rough intention of the functions and class behaviours of the system.

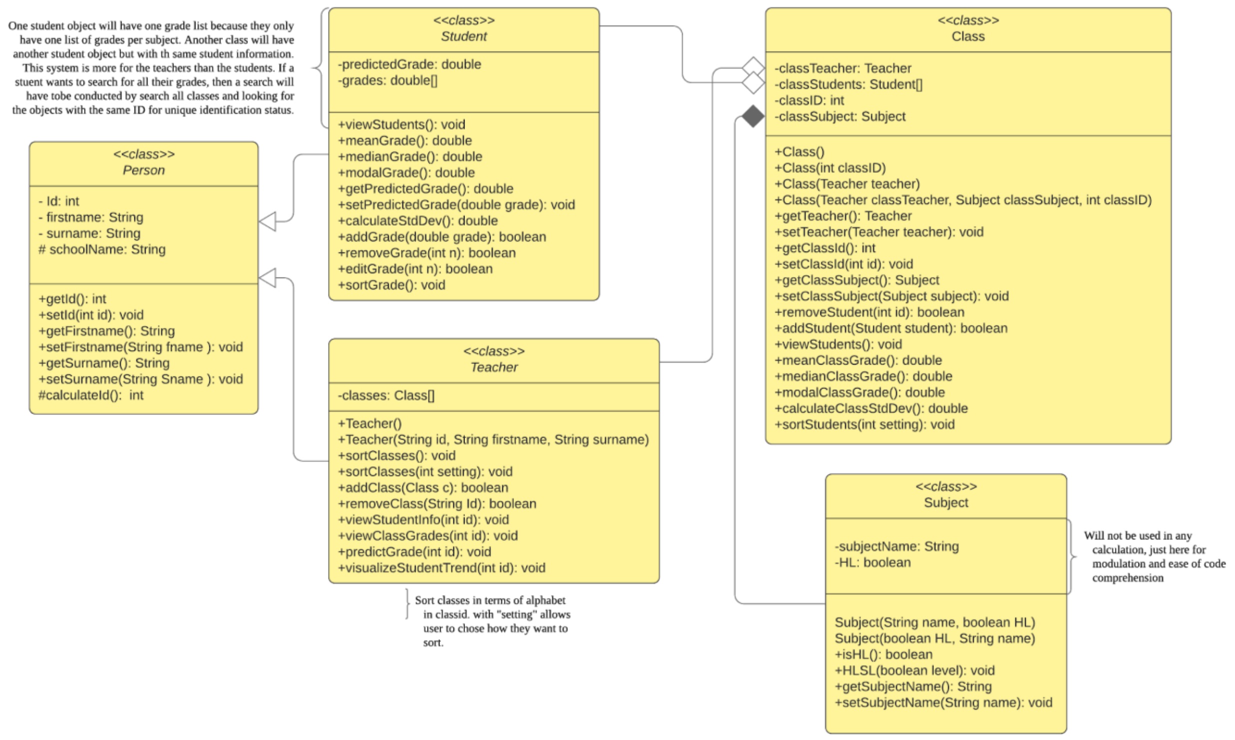
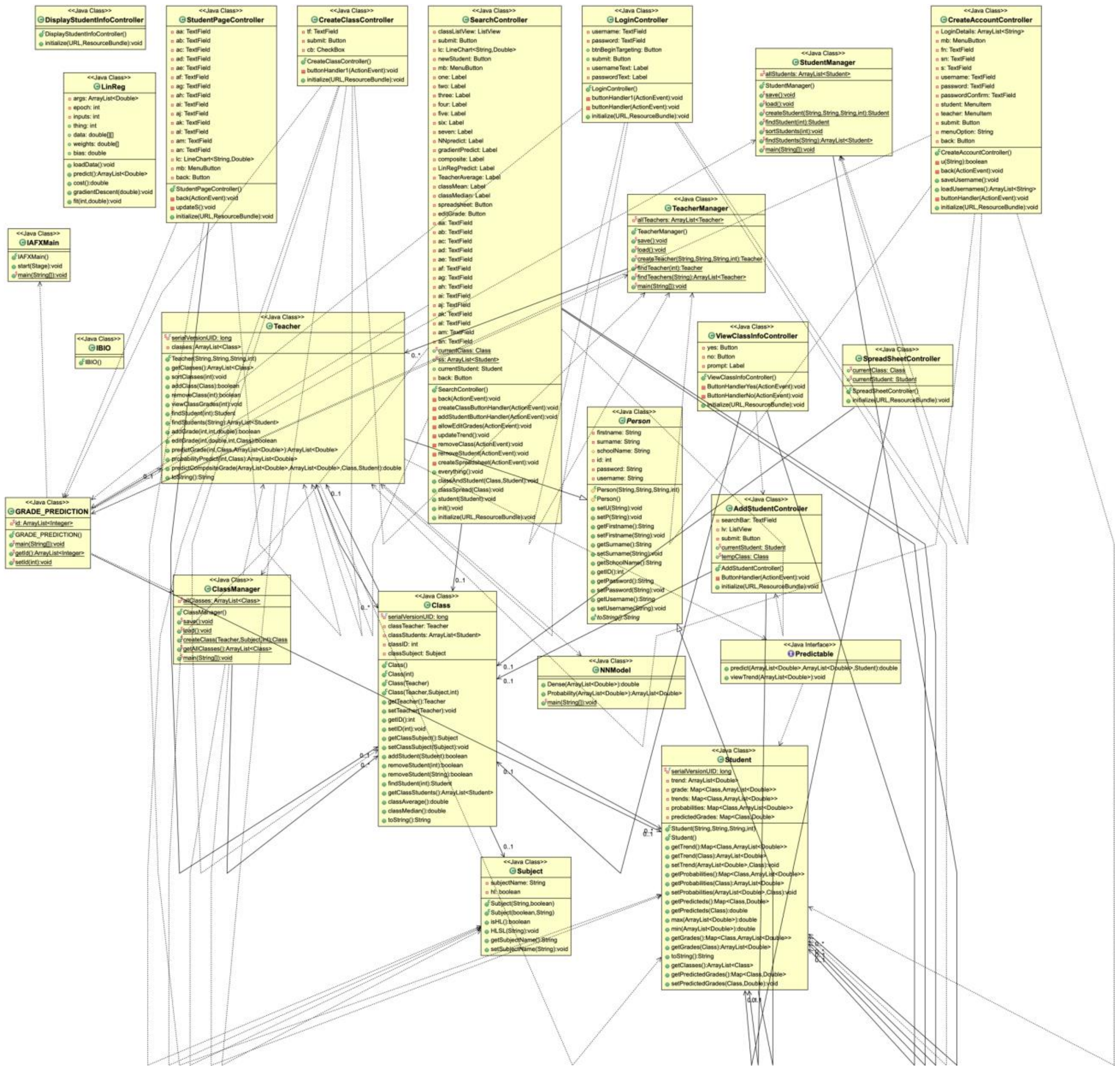

One student object will have one grade list because they only have one list of grades per subject. Another class will have another student object but with th same student information. This system is more for the teachers than the students. If a stuent wants to search for all their grades, then a search will have tobe conducted by search all classes and looking for the objects with the same ID for unique identification status.
<<class>>
Student
-predictedGrade: double
-grades: double[]
+viewStudents(): void
+meanGrade(): double
+medianGrade(): double
+modalGrade(): double
+getPredictedGrade(): double
+setPredictedGrade(double grade): void
+calculateStdDev(): double
+addGrade(double grade): boolean
+removeGrade(int n): boolean
+editGrade(int n): boolean
+sortGrade(): void
<<class>>
Class
-classTeacher: Teacher
-classStudents: Student[]
-classID: int
-classSubject: Subject
+Class()
+Class(int classID)
+Class(Teacher teacher)
+Class(Teacher classTeacher, Subject classSubject, int classID)
+getTeacher(): Teacher
+setTeacher(Teacher teacher): void
+getClassId(): int
+setClassId(int id): void
+getClassSubject(): Subject
+setClassSubject(Subject subject): void
+removeStudent(int id): boolean
+addStudent(Student student): boolean
+viewStudents(): void
+meanClassGrade(): double
+medianClassGrade(): double
+modalClassGrade(): double
+calculateClassStdDev(): double
+sortStudents(int setting): void
<<class>>
Person
- Id: int
- firstname: String
- surname: String
# schoolName: String
+getId(): int
+setId(int id): void
+getFirstname(): String
+setFirstname(String fname ): void
+getSurname(): String
+setSurname(String Sname ): void
#calculateId(): int
<<class>>
Teacher
-classes: Class[]
+Teacher()
+Teacher(String id, String firstname, String surname)
+sortClasses(): void
+sortClasses(int setting): void
+addClass(Class c): boolean
+removeClass(String Id): boolean
+viewStudentInfo(int id): void
+viewClassGrades(int id): void
+predictGrade(int id): void
+visualizeStudentTrend(int id): void
Sort classes in terms of alphabet in classid. with "setting" allows user to chose how they want to sort.
<<class>>
Subject
-subjectName: String
-HL: boolean
Subject(String name, boolean HL)
Subject(boolean HL, String name)
+isHL(): boolean
+HLSL(boolean level): void
+getSubjectName(): String
+setSubjectName(String name): void
Will not be used in any calculation, just here for modulation and ease of code comprehension

Figure 4: UML Diagram with intended functions to code

*Post Development UML Diagram to show the development of the system:*



3. **Use-case Diagram:** This diagram outlines the basic understanding of the user's interaction with the system.

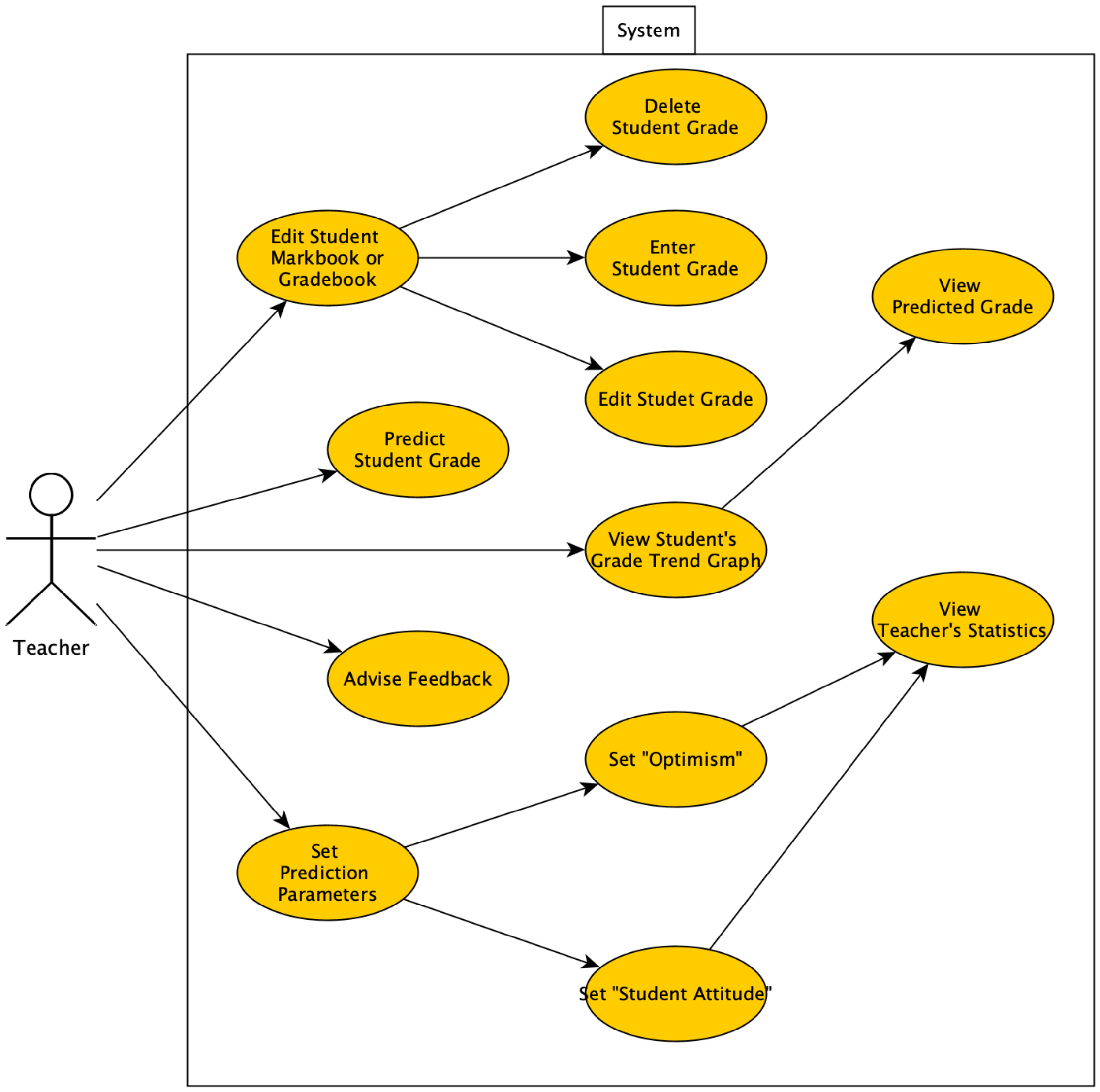


Figure 5: Use-Case Diagram of Clients System

4.**Data Dictionary:** An appendix of variables name, type, and a short description of their intended use in the system. It is intended to improve technical documentation of the new system to increase the readability and comprehension of the system.

| **Attribute** | **Data Type** | **Modifier** | **Description** |
|---|---|---|---|
| | | *Student* | |
| predictedGrade | Double | private | This is the predicted grade of the student for a specific subject. This ranges from 1 to 7. |
| | | *Class* | |
| classID | Integer | private | This is a unique identifier for the class. Using classID, the software will be able to precisely identify any class. |
| classTeacher | Teacher | private | Every class must have a teacher. This is an instance of the Teacher class |
| classStudents | ArrayList<Student> | private | Every class must have a student for a teacher to teach. This is a list of students |
| classSubject | Subject | private | The class has to have a subject that the teacher will teach, and the students will learn, This is an instance of the *Subject* class |
| | | *Subject* | |
| subjectName | String | private | The subject will have a name, for example, “Mathematics” |
| HL | boolean | private | In the IB, there is a distinction between a Higher Level SUbject (HL) or a standard level subject(SL). If *HL* is true, then the subject is a Higher Level subject; if *HL* is false, then the subject is a standard level subject. |
| | | *Teacher* | |
| classes | ArrayList<Class> | | A teacher can teach multiple classes. Therefore, a teacher can access all students in each class they teach. |

| *Person* | | | |
|---|---|---|---|
| Id | int | private | Each Person can be identified through a specific, unique 5 digit ID. |
| firstname | String | private | Every person has a first name and a surname. |
| surname | String | private | |
| schoolName | String | protected | This is the name of the school the student is in. Although it doesn’t serve a purpose, this variable was added for completeness of the system. |

Figure 6: Table showing the various variables used in each class in the client’s system

## 5.**Pseudocode**:

*Sorting Algorithms*

Each class will have an ArrayList of students, instead of an array because class sizes are variable and Arraylists will save space. The Students will be sorted, in this array, by their ID's. As they are integers, sorting will require an algorithm efficient at sorting integers from lowest to highest. An efficient algorithm to do this ( an already formed array ) is the insertion sort.

In the algorithm:

STUDENT = ArrayList of students in a class being sorted
NUMBER_OF_ELEMENTS = number of elements in the array
SMALLEST_ELEMENT = area for holding the smallest element found in that pass
CURRENT_SMALLEST_POSITION = the value of the current position in which to place the smallest element
I = index for outer loop
J = index for inner loop.

```
function selectionSort(STUDENT): void
    CURRENT_SMALLEST_POSITION = 1
    loop while CURRENT_SMALLEST_POSITION <= (NUMBER_OF_ELEMENTS – 1)
        I = CURRENT_SMALLEST_POSITION
        SMALLEST_ELEMENT = STUDENT.get(I)
        J = I + 1
        loop while J <= NUMBER_OF_ELEMENTS
            if STUDENT.get(J) < SMALLEST_ELEMENT then
                I = J
                SMALLEST_ELEMENT = STUDENT.get(J)
            end if
            J = J + 1
        end loop
        STUDENT.set(I,STUDENT.get(CURRENT_SMALLEST_POSITION)
        STUDENT.set(CURRENT_SMALLEST_POSITION, SMALLEST_ELEMENT)
        add 1 to CURRENT_SMALLEST_POSITION
    end loop
end function
```

1. *adapted from (Robertson, Lesley Anne.)*

## *Searching Algorithms*

Assuming that the array of students is sorted in ascending order of ID, to search for a specific student, the teacher will input the ID of the student, and the searching algorithm will search for the student with the ID. To aid in the process of efficiency, and ease of processing power, a Binary Search. The efficiency of this algorithm is $O(log_2 n)$. This is more efficient than other search algorithms such as the Linear/Sequential search, with an efficiency of $O(n)$.

In the algorithm:

STUDENT_COLLECTION = array or list to be searched
IS_FOUND = flag containing true if found, false otherwise
PLACE = the index of the key element in the collection, or -1
LENGTH = number of items (size) of the COLLECTION
LOW = lower index or left index of a sub-array where the key is searched
HIGH = highest index or right index of a sub-array where the key is searched

Assume that the contents of the COLLECTION and LENGTH have already been established. COLLECTION must be sorted in ascending order (small to large). This is a modified version of the binary search that, instead of returning the index of the elements, will return the STUDENT object in that index.

```
function binarySearch(STUDENT_COLLECTION, STUDENT_ID) : Student
        set IS_FOUND to false
        set LOW to 0
        set HIGH to LENGTH
        set PLACE to -1
        loop while not FOUND and (LOW < HIGH)
                set INDEX to (LOW + HIGH) / 2
                if STUDENT_COLLECTION.get(INDEX)== STUDENT_ID then
                        set IS_FOUND to true
                        set PLACE to INDEX
                        exit loop
                else
                        if STUDENT_ID < STUDENT_COLLECTION.get(INDEX) then
                                set HIGH to INDEX – 1
                        else
                                set LOW  to INDEX + 1
                        end if
                end if
        end loop
        return STUDENT_COLLECTION.get(PLACE)
end function
```

*adapted from (Robertson, Lesley Anne.)*

6.**Flowcharts**: These flowchart helps visualise the different parts of the system and their relationships.

*Predictive Algorithm*

To form a predicted grade, to graph all grades, find the line of best fit, and project the trend line into future timelines, would be a somewhat accurate representation of predicted grades. This is so because the predicted grade would take into account all grades before it, and the accuracy of the prediction model would increase as the number of grades entered into the system increased. However, this has various limitations.

One severe limitation of such prediction is that a student's attitude can change at any time during the I.B.' journey'. It is possible, although highly unlikely, for a student to 'jump' from obtaining constant 3's in a subject to possibly a 5. Furthermore, unless told otherwise, the linear regression model will take "outliers" ( tests where the student dropped grade significantly ) into consideration, and thus lowering the grade even though that momentary dip in the student's standard was only 'momentary,' and not an accurate representation of the students real level in the subject. A layer of prediction that could be appended to the model can be for the model to take into account previous students with similar trends in their grades. A pattern prediction model that views how other students with the same trend in grades performed relative to their existing standard.  If many students follow the same pattern in the grade, then the final result on the grade may be in proportion with the trend they displayed throughout the course. For instance, a student with grade pattern 3,4,3,4,3,4 who then achieves a final grade 4; A student who follows a similar trend 5,6,5,6,5,6, would then likely obtain a 6, because the trend is similar. This entire prediction will, from now on, be known as '***Prediction 1***.' This is a neural network coded in python. It is a sequential model that takes an input layer of 14 neurons. After Various dense hidden layers neurons each to fit the data, the output is a regression output, which outputs a single grade from 1 to 7.9.

However, the Line of Best Fit Model is still good to see student trends, so we have two layers in the prediction. A 'L.O.B.F.' model (linear regression) that is a straight line through all data points. This will henceforth be known as '***Prediction 2***.'

Furthermore, according to my client Mr. Baker, teachers aren't allowed to predict a grade higher than the maximum grade the student obtained, so my model will have a maximum grade that can be predicted.

Note: *These are only algorithms and processes for the prediction of the number grade, and does not include the display of the 'trend-line graph.'*

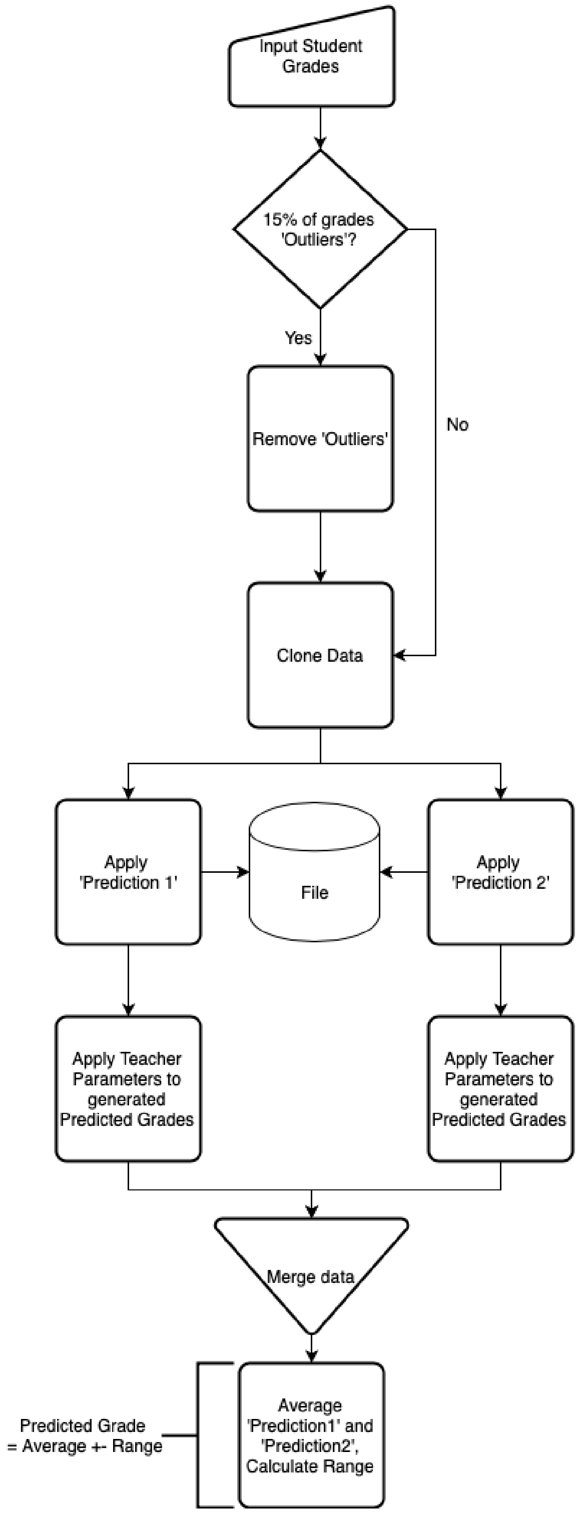

Input Student Grades
15% of grades 'Outliers'?
Yes
No
Remove 'Outliers'
Clone Data
Apply 'Prediction 1'
File
Apply 'Prediction 2'
Apply Teacher Parameters to generated Predicted Grades
Apply Teacher Parameters to generated Predicted Grades
Merge data
Average 'Prediction1' and 'Prediction2', Calculate Range
Predicted Grade = Average +- Range

Figure 7: Flow chart of the grade prediction process

_Validation_: These are flowcharts of the validation of the input of student grades and the input of names into the system.

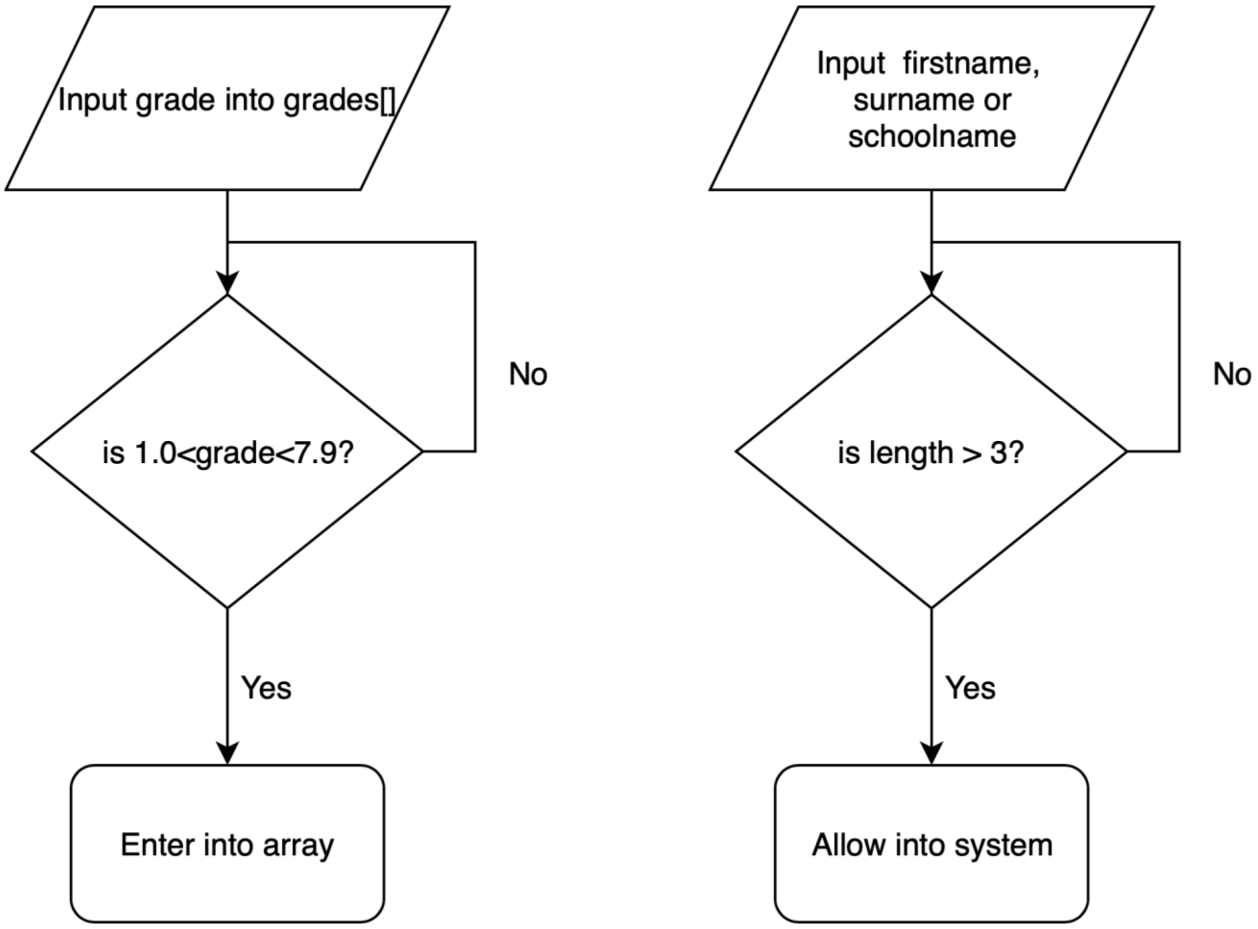


Figure 8: Flowcharts showing the validation process of grades and names

**in the final product, this validation was changed from 3 to 1 at the clients request*

7.**Test Plan**:

**Class: Person**
Id: int

| **Type of Validation** | **Test Data Type** | **Input Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| **Range Check( *0<x<99999 )*** | **Normal** | **57538** | Accept value | ✓ |
| | **Extreme** | **982376; -5** | Reject value and input again | ✓ |
| | **Abnormal** | **“6263”** | Reject value and input again | ✓ |

**if ID exists then ID should be input again. ID should be unique → ✓*

firstname, surname, schoolName: String

| **Validation** | **Test Data Type** | **Test Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| **Length Check (>1)** | **Normal** | **“john”** | Accept value | ✓ |
| | **Extreme** | **“Ab” or “Sjohn123”** | Reject value and input again | ✓ |
| **Format Check (Alphabet only)** | **Abnormal** | **1234** | Reject value and input again | ✓ |

**Class: Class**
classTeacher: *Teacher*

| **Type of Validation** | **Test Data Type** | **Input Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| | **Normal** | **Teacher = !null** | Accept value | ✓ |
| **Presence Check** | **Extreme** | **null** | Must input before processing | ✓ |
| | **Abnormal** | **null** | Break out of method/ Halt further processing/ Must input again before processing | ✓ |

classSubject: *Subject*

| **Type of Validation** | **Test Data Type** | **Input Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| **Presence Check** | **Normal** | **classSubject != null** | Accept value | ✓ |
| | **Extreme** | **null** | Must input before processing | ✓ |
| | **Abnormal** | **null** | Break out of method/ Halt further processing/ Must input again before processing | ✓ |

**Subject**
subjectName: String

| **Validation** | **Test Data Type** | **Test Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| **Length Check>1** | **Normal** | **"Mathematics 02"** | Accept value | ✓ |
| | **Extreme** | **""** | Reject value and input again | ✓ |
| | **Abnormal** | **123** | Reject value and input again | ✓ |

subjectName: String

| **Validation** | **Test Data Type** | **Test Data** | **Expected Output** | **Test Passed ✓ Test Failed X** |
|---|---|---|---|---|
| **Format Check (Alphabet including numbers)** | **Normal** | **"Mathematics 02"** | Accept value | ✓ |
| | **Extreme** | **"$$$$$Mathematicsj12"** | Reject value and input again | ✓ |
| | **Abnormal** | **123** | Reject value and input again | ✓ |

**Student**
grades[]: double

| **Validation** | **Test Data Type** | **Test Data** | **Expected Output** | **Test Passed ✓ est Failed X** |
|---|---|---|---|---|
| **Range Check; 1~7.9** | **Normal** | **7.2** | Accept value | ✓ |
| | **Extreme** | **0.8 ; 8.2** | Reject value and input again | ✓ |
| | **Abnormal** | **seven point three** | Reject value and input again | ✓ |

Predicted Grade: double

| Validation | Test Data Type | Test Data | Expected Output | Test Passed ✓ Test Failed X |
|---|---|---|---|---|
| Type check; Range Check Double: (1.0~7.9) | Normal | 7.2 | Accept value | ✓ |
| | Extreme | 0; 9 | Reject value and input again | ✓ |
| | Abnormal | seven point three; | Reject value and input again | ✓ |

Figure 9: Array of tables showing validation of all variables in the Client's system

8.**GUI Draft Layout:**

This is a draft layout of what the main screen would look like. In a specific class a teacher teaches, a teacher can select the student (click on him/her/other) and view only relevant data. This diagram is only a brainstorm and is subject to change. The graph in the middle is supposed to represent the "trend graph" of the students grades against time. The purpose of this diagram is to ease implementation and development of the system as a plan can be followed to do so.

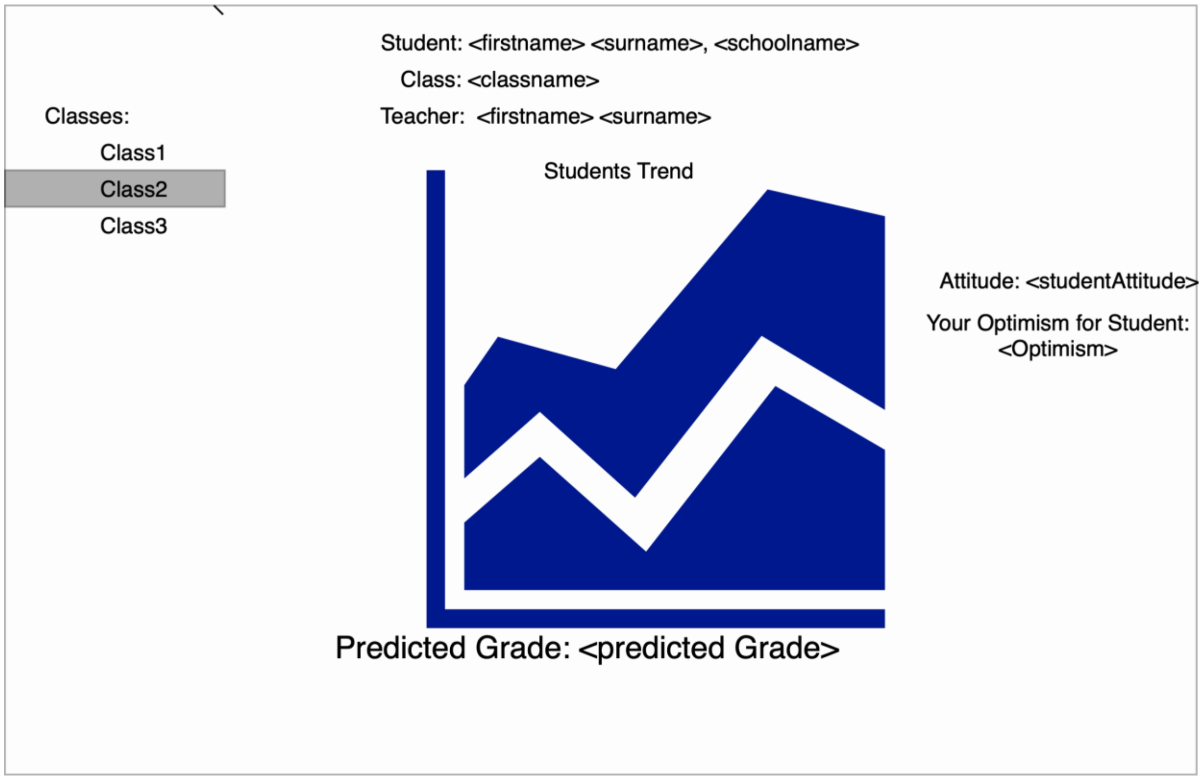

Student: <firstname> <surname>, <schoolname>
Class: <classname>
Teacher: <firstname> <surname>
Classes:
Class1
Class2
Class3
Students Trend
Attitude: <studentAttitude>
Your Optimism for Student: <Optimism>
Predicted Grade: <predicted Grade>

Figure 10: DRAFT GUI of the main page of the system

9.**Linear Regression Mathematics:**

Linear Regression, as mentioned as **Prediction 2** in my code is an application of the Machine Learning algorithm. The following is to brainstorm the theory in a format that is relatively easy to understand to aid the process of developing the algorithm (in java, which is rare) at a later stage. Linear Regression is essentially finding the line of best fit through a set of data points. For instance: the data points below:

| $X$ | $Y$ |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 6 |
| 4 | 8 |
| 5 | 10 |

In the table above, let the first data set be denoted at $X_1, Y_1 = 1{,}2$, and $X_2, Y_2 = 2{,}4$ and so on. As seen, if a line $h(X_i) = mX_i + c$ represent the straight line that runs through all of the above data sets, then the following statements must be true:

$h(1) = m(1) + c = 2$,$h(2) = m(2) + c = 4$, $h(3) = m(3) + c = 6$and so on for all data points.

At the very beginning of the algorithm, the variables $m$and $c$begin as random numbers, say m = 0.63475 and c = 0.23746, and therefore, with input $X_i$, the prediction will output a random number: $h(X_i)$ **= random number.**

The essence of the code is to minimize the absolute distance between the prediction $h(X_i)$ and the actual value $Y_i$. Therefore, we have to minimize the expression $(|h(X_i) - Y_i|)^2$. This is known as the cost. The reason that the difference is squared is to account for negative numbers. The goal is to bring $(|h(X_i) - Y_i|)^2$as close to 0 as possible, representing that the prediction $h(X_i)$and the actual value $Y_i$ are the same number.

However, we have to compute this cost expression for every data set, because the prediction line runs through every data set. So the total cost is equal to $(|h(X_1) - Y_1|)^2 + (|h(X_2) - Y_2|)^2 + \ldots.$. This sum continues for all data sets that are being continued. In terms of the numbers in the table, the expression is $(|h(1) - 2|)^2 + (|h(2) - 4|)^2 + (|h(3) - 6|)^2 + (|h(4) - 8|)^2 + (|h(5) - 10|)^2$, remembering that $h(1)$is the prediction and $Y$is the actual value that has been given to the model, and that we want that entire expression to be as close to 0 as possible.

So: $Cost = \frac{1}{v}\sum_{i=0}^{v} \quad [(h(X_i) - Y_i)^2] = \frac{1}{v}\sum_{i=0}^{v} \quad [(\,(m(X_i) \quad - c) - Y_i)^2]$ .

As complex as t looks, it is simply the operation that was conducted above summing all the costs from all inputs and outputs, where only this time, V denotes the number of data sets, in our case, 5, and $\frac{1}{v}$denotes that the average cost of all data sets are being computed.

Now, the question to ask, is what values of m and $c$ minimize the cost: get the prediction $(h(X_i)$ as close to the actual value, $Y_i$, as possible. In Calculus, to find the minimum cost with respect to the variables are given by the expressions.
$\frac{\partial\partial J}{\partial m}$and $\frac{\partial\partial J}{\partial c}$.To calculate these expressions:

$$\frac{\partial\partial J}{\partial m} = \sum \quad \frac{\partial\partial J}{\partial h(X_i)} \cdot \frac{\partial\partial h(X_i)}{\partial m}$$

$$\frac{\partial\partial J}{\partial h(X_i)} = 2(h(X_i) - Y_i); \frac{\partial\partial h(X_i)}{\partial m} = \frac{\partial\partial}{\partial m}(mX_i + c) = X_i$$

$$\frac{\partial\partial J}{\partial m} = \frac{1}{v}\sum_{i=0}^{v} \quad [2(h(X_i) - Y_i)X_i]$$

**In english,**$\frac{\partial\partial J}{\partial m} = 2(h(X_i) - Y_i)X_i$ **means: when** $m$**increases,** $J$**will increase by** $2(h(X_i) - Y_i)X_i$ **amount**

$$\frac{\partial\partial J}{\partial c} = \sum \quad \frac{\partial\partial J}{\partial h(X_i)} \cdot \frac{\partial\partial h(X_i)}{\partial c}$$

$$\frac{\partial\partial J}{\partial h(X_i)} = 2(h(X_i) - Y_i); \frac{\partial\partial h(X_i)}{\partial c} = \frac{\partial\partial}{\partial c}(mX_i + c) = 1$$

$$\frac{\partial\partial J}{\partial c} = \frac{1}{v}\sum_{i=0}^{v} \quad [2(h(X_i) - Y_i) \cdot 1]$$

**In english,**$\frac{\partial\partial J}{\partial c} = 2(h(X_i) - Y_i) \cdot 1$ **means: At any point, when** $c$ **increases,** $J$**will increase by** $2(h(X_i) - Y_i) \cdot 1$ **amount.**

To minimize the cost, $J$,we have to ask the question : At any point, what value of $c$&$m$will make $J$not change?

To find this, the following algorithm is conducted

```
REPEAT UNTIL CONVERGENCE{
    $m = m - \alpha \frac{\partial\partial J}{\partial m}$
    $c = c - \alpha \frac{\partial\partial J}{\partial c}$
}
```

By subtracting the gradients, they are being shaped to fit the data better. After they are updated, a new cost is calculated and the cycle continues. As m and c continuously get updated, they start to converge towards a value because their costs get closer to 0. This cycle continues until the cost converges to 0, this typically happens after 10000 updates, answering the question “at any point, what value of $c\&m$will make $J$not change?”

In the case of the values in the table above, m and c will probably converge to 1.9999999999999 and 0.00000001 respectively.

I plan for my algorithm to embrace a similar nature, where the grades of a student are the $Y$and the $X$ranges from {0,1,2,3,4,5,...13} (14 values). After the values for m and c are found, then I simply plug in $X$ = 14 to find the next value on the straight line formed from the past data.
I give the algorithm $(X_0, Y_0), (X_1, Y_1)..., (X_{13}, Y_{13})$ , and I predict using $X_{14}$to find the predicted grade $Y_{14}$. For instance, if a student attains the following grades {2.5,2.5,3.5,5.5,5.5}, the input data would look like this

| $i$ | $X_i$ | $Y_i$ |
|---|---|---|
| 0 | 0 | 2.5 |
| 1 | 1 | 2.5 |
| 2 | 2 | 3.5 |
| 3 | 3 | 5.5 |
| ... | ... | ... |
| 13 | 13 | 5.5 |
| 14 | 14 | Predicted Grade |

## Changes to the System After Development

*Made the system more fully connected*
To do this, I included more aggregation. Each class has a copy of every other class, and each object that is related to other classes will have some reference to it. After this improvement, it is possible to navigate through the entire system with any single object. Of course, appropriate data hiding and encapsulation prevent unwanted access to private variables, and my code only utilizes the aforementioned connections when necessary. Nonetheless, I still abide by my new method because it allows for easier back-end navigation: loading and storing objects using serialization, using variables from other objects instantiated from other classes, and creating the grade prediction algorithm in the *Teacher.java* class all became easier. This was implemented through aggregation, and adding ArrayLists and HashMaps of other classes in host classes. For instance, the Student class has a HashMap that links a Class to an array of grades. Each Class has its own individual ArrayList of students, which can be used to access other students including the original one, and a Class Teacher. The Teacher has an ArrayList of Classes( objects ). This made the system more connected.

*Method of prediction*

Removed Teacher Attributes)
I removed the teacher attributes. Although they added a crucial human element to the prediction process, I felt that it was an unnecessary layer. The purpose of the software is to provide a fair basis of predicting grades simply on numbers. If my client has the option to allow his emotions to increase and decrease.

Added a new linear Regression model)
The first Linear Regression Model finds  the straight line of best fit through the grades provided by the teacher. It will then plot the next point on the line as the prediction. In the product, it is denoted by the orange line in the graph. This new Linear regression algorithm uses the same class, but the input data is different. Instead of an array of all the grades, the new algorithm intake the change in grades, essentially the gradients. The algorithm predicts the next change in the students' grades they are likely to attain. Furthermore, I also made both linear regression models account for outliers. If the change from the previous to the current grade is more than 1.5 grade levels, the algorithm will diminish the gradient by 30%.

The second implementation of the model is to predict the next change in grades. For instance, if a student attains the following grades  {2.5,2.5,3.5,5.5,5.5}, the input data would look like this (Using the same notation as the example in planning above).

| $i$ | $X_i$ | $Y_i$ |
|---|---|---|
| 0 | 0 | 2.5-2.5 = **0** |
| 1 | 1 | 3.5-2.5 = **1** |
| 2 | 2 | 5.5-3.5 =**2** |
| 3 | 3 | 5.5-5.5 = **0** |

The prediction will predict the next change. In this case, it will probably predict a change of 1, leading the algorithm to predict that the next grade is 5.5+1, which is 6.5.

Probabilities)
This is a neural network coded in python. I added this as a new prediction model. It is a sequential model that takes an input layer of 14 neurons. After Various dense hidden layers of 16, 32, 128, 7 neurons each to fit the data, The output layer is a softmax layer, which outputs the probabilities that the set of inputted grades will be either 1,2,3,4,5,6,7.

*Added Student Page)*
To add completeness to the system, and to allow students to see their progress in each class. I did this by creating a difference between a Teacher Account and a Student Account using inheritance.

*Changed the Prediction algorithm for the composite predicted grade.*
There are various components of the composite predicted grade. L.O.B.F. Trend; Gradient Trend; TA; Neural Network; and Probabilities. The LONG Trend is the linear regression model mentioned in my planning. The Gradient Trend is the gradient trend discussed above. The T.A is the average of all semester exams provided by the system, and the NN is the grade generated from the neural network prediction. I added a feature where the teacher can view the probability that the student will attend each grade level, e.g. 50% 7 and 30% 6, etc etc. This was also accomplished using a neural network. This is how the composite was calculated:

First, I calculate the expected value of probability function of the neural network probabilities.
$E(X) = \sum \quad XP(X)$.
For instance, if the probability is

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0.15 | 0.75 | 0.1 |

According to this, the probability predicted that the student has 15% chance of attaining a 5, an 75% chance of attaining a 6, and a 10% chance of attaining a 7. However, although the student is most likely to get a 6, it isnt sa 100% probability. So I found the expected value as it is more accurate. Here, the student is expected to get :$0 \times 1 + 0 \times 2 + 0 \times 3 + 0 \times 4 + 0.15 \times 5 + 0.75 \times 6 + 0.1 \times 7 = 5.95$
The prediction is then calculated from the average of LONG Trend; Gradient Trend; TA;NN;, then this is averaged with the Expected value (5.95).

**Extensions and modifications (From D)**

| Extension or Modification | Location | Development |
|---|---|---|
| Change Prediction Algorithm | 1 - PGA.py; PGA1.py<br>2 - NNModel.java<br>3 - Teacher.java | 1 - These python files contain the model itself. The connections, and the numbers of neurons in the model at each layer can be altered here by changing the numbers. This could be to mold the model to fit the training data better, allowing for a more accurate prediction.<br><br>2 - This Class contains the implementation of the model in java. It acts as a bridge between the .json and .h5 files created by the python model and the system.<br><br>3 - Here, there is a function predictGrade() that contains all of the predictions. The data can be preprocessed in a different way, like it was for the gradient prediction, and a new prediction model can be created by creating another installation of the predictable.java interface. |
| Dependencie s | | They can be updated to allow for maximum use and functionality. The models, in fact, only work with version beta 6 or newer of the dependency |
| Change GUI | All Controller Classes. | In the SearchController.java class, the method updateTrend() is a primary method that updates the Teacher screen.. It controls running the prediction algorithms and updates the Labels, Scenes, and Graphs to update the correct information in live time. |
| Change Grade Range | 1 - SearchController.j ava<br>2 - PGA.py; PGA1.py<br>3 - NNModel.java<br>4 - Teacher.java | The grade range, for instance, non-IB numerical grades specific to other schools, or letter grades (A,B,C,D…) is a viable modification that can be coded into the system because schools have different grading systems. In this case, the Validation, currently 1.0<x<7.9, can be changed in SearchController.java . Also, for the prediction algorithm, letter grades can be calculated by inputting numbers such as 4.0 to represent a grade A, and 3.0 to present B. The neural network regression outputs will be grades between 1.0 and 4.0, instead of 1.0 and 7.9. Say, 3.0-3.5 is B~B+, and 3.8~4.0 is an A: these ranges can be defined using simple if-statement-blocks. These can be changed in the Teacher.java class when data is entered into the neural network, or NNModel.java, when the deeplearning4j implementation of the neural networks predict. |