

## **Harnessing Peer Power: A Novel Approach to Auto Answer Evaluation in Education**

In large classes and MOOCs, the traditional method of manual evaluation by teachers and TAs can be overwhelming due to the sheer volume of student responses. This often leads to delays in teaching / Quality of teaching, which can hinder student learning and engagement. To address this challenge, we propose a peer evaluation system for auto answer evaluation that distributes the evaluation workload among students.

Key Components of the Peer Evaluation System:

1. Auto Evaluation Mechanism: Develop an automated system that can evaluate student answers based on predefined criteria. This system should be capable of handling various types of questions, including multiple-choice, short answer, and essay questions.
2. Peer Evaluation Framework: Implement a peer evaluation framework where students are assigned a set of answers from their peers to evaluate based on the same criteria used by the automated system. **Each answer should be evaluated by multiple peers to ensure fairness and accuracy.**
3. Incentive Structure: Provide incentives for students to participate in the peer evaluation process. This could include bonus points, recognition, or other rewards that motivate students to take the task seriously.
4. Quality Control Mechanisms: Implement quality control mechanisms to ensure the accuracy and reliability of peer evaluations. This could include cross-checking evaluations by multiple peers and allowing students to appeal the evaluations.
5. Feedback Loop: Establish a feedback loop where students receive feedback on their evaluations and have the opportunity to learn from their peers' feedback on their own answers. This promotes a culture of continuous improvement and peer learning.

## Benefits of the Peer Evaluation System:

1. **Reduced Workload:** By distributing the evaluation workload among students, the burden on teachers and TAs is significantly reduced, allowing them to focus on other aspects of teaching and learning.
2. **Faster Feedback:** The peer evaluation system can provide faster feedback to students, enabling them to identify and address their mistakes more quickly, leading to improved learning outcomes.
3. **Promotes Critical Thinking:** Evaluating their peers' answers can help students develop critical thinking and analytical skills, as they are required to assess the quality of the answers based on predefined criteria.
4. **Encourages Engagement:** The incentive structure and peer interaction involved in the evaluation process can increase student engagement and motivation, leading to a more active learning environment.

Overall, implementing a peer evaluation system for auto answer evaluation in large classes and MOOCs can lead to more efficient and effective evaluation processes, benefiting both students and teachers.

## 1. Introduction:

Traditional teaching methods often involve passive or active learning, which can lead to an increased workload for teachers and TAs, particularly in large classes with more than 100 students. To alleviate this burden, we propose a hierarchical teaching approach that distributes the workload evenly among teachers, TAs, super peers, and peers (students). This approach not only reduces the workload on instructors but also promotes collaborative learning among students.

Learning Management Systems (LMS) are widely used for assignments, quizzes, and short-answer questions in educational settings. However, these systems typically require the presence of teachers or TAs for evaluation, limiting their effectiveness as

true auto evaluation systems. Peer review and feedback systems have been recognized in literature as effective methods for reducing the evaluation burden on instructors. In this paper, we focus on the implementation of a peer evaluation system for auto answer evaluation.

In our proposed peer evaluation system, students act as evaluators for their peers' answer scripts. They receive an answer scheme from the teacher or TA and evaluate the answer scripts accordingly. The evaluation process is autonomous, meaning that students do not know who evaluated their answers, and vice versa. Additionally, **each answer sheet is evaluated multiple times by different peers to ensure accuracy and fairness in evaluation.**

The peer evaluation system presented in this paper offers a novel approach to auto answer evaluation in large classes and MOOCs. By leveraging the collective intelligence of students, this system effectively distributes the evaluation workload, reducing the burden on teachers and TAs. However in this report we have taken the help of both Teachers and Teaching Assistants to compare and justify the results obtained. Further research could focus on implementing and evaluating this system in real-world educational settings to assess its effectiveness and scalability.

## **2. Literature Review:**

### **2.1 LearnEval Peer Assessment Platform: Iterative Development Process and Evaluation (Gabriel Badea and Elvira Popescu, Senior Member, IEEE):**

"LearnEval Peer Assessment Platform: Iterative Development Process and Evaluation" by Gabriel Badea and Elvira Popescu, Senior Member, IEEE, provides a comprehensive overview of the development, features, and evaluation of LearnEval, a peer assessment platform designed for educational settings. The paper outlines the iterative development process that was undertaken to create the platform, emphasizing the various stages and methodologies utilized to enhance its functionality and user-friendliness.

The initial design of LearnEval aimed to simplify the peer assessment process, ensuring fairness, anonymity, and reliability in evaluations. Challenges encountered during development, such as algorithm refinement for automatic answer evaluation and feedback generation, are discussed. Integration of machine learning algorithms into the platform is highlighted as a key strategy to improve its capabilities.

The paper also delves into the evaluation of LearnEval, detailing the methodologies used to assess its effectiveness. This includes user surveys, performance metrics analysis, and comparative studies with existing peer assessment tools. Results from these evaluations indicate that LearnEval has a positive impact on student learning outcomes and can streamline the assessment process in educational settings.

Overall, the paper presents LearnEval as a robust and innovative peer assessment platform. Its iterative development approach and thorough evaluation process contribute valuable insights into the design, implementation, and evaluation of similar platforms in educational environments.

## **2.2 Incentive Design in Peer Review: Rating and Repeated Endogenous Matching (Yuanzhang Xiao, Florian Dorfler, and Mihaela van der Schaar, Members/Fellows, IEEE):**

The paper "Incentive Design in Peer Review: Rating and Repeated Endogenous Matching" by Yuanzhang Xiao, Florian Dorfler, and Mihaela van der Schaar, published in IEEE Transactions on Network Science and Engineering, addresses the challenges of traditional peer review systems and proposes a novel incentive mechanism to improve the quality and integrity of the review process.

The authors highlight the critical role of peer review in academic publishing and other domains for ensuring the validity and quality of submissions. However, they note several shortcomings in traditional peer review, including biased reviews, strategic behavior by reviewers, and a lack of sufficient incentives for reviewers to provide thoughtful and accurate evaluations.

To address these challenges, the authors propose a new incentive mechanism that combines two key components: rating and repeated endogenous matching. In their system, reviewers are not only asked to provide a binary decision (accept/reject) for a submission but also to rate the submission on a continuous scale. Reviewers are then matched repeatedly with the same submission over multiple rounds, allowing them to update their ratings based on feedback from other reviewers. This repeated matching incentivizes reviewers to provide accurate and thoughtful evaluations, as their reputations are tied to the quality of their ratings.

The paper discusses the theoretical properties of the proposed mechanism, including its convergence properties and its ability to incentivize truthful reporting. The authors also conduct simulations to demonstrate the effectiveness of their approach compared to existing mechanisms. The results show that the proposed mechanism improves the quality of reviews and incentivizes honest behavior among reviewers.

Overall, the paper provides a comprehensive framework for designing incentives in peer review systems, aiming to enhance the quality and integrity of the review process in academic publishing and beyond.

### **2.3 A Framework for Teaching Evaluation (Divya Nalla, Department of Computer Science, Nalla Malla Reddy Engineering College):**

The paper proposes a comprehensive framework for evaluating teaching effectiveness, which could be beneficial for educators, administrators, and policymakers. The framework likely includes various components such as:

1. **Objective Setting:** Defining clear objectives for teaching evaluation, ensuring alignment with the educational goals of the institution or program.
2. **Metrics Selection:** Identifying relevant metrics or criteria for evaluating teaching effectiveness, which could include student performance, student feedback, peer evaluations, and self-assessment by instructors.
3. **Data Collection:** Describing methods for collecting data on teaching effectiveness, such as surveys, classroom observations, and analysis of student outcomes.

4. Data Analysis: Providing a systematic approach for analyzing the collected data to assess teaching effectiveness objectively.
5. Feedback Mechanism: Implementing a feedback mechanism to communicate evaluation results to instructors, allowing them to improve their teaching practices.
6. Continuous Improvement: Emphasizing the importance of continuous improvement in teaching based on evaluation results and feedback.
7. Implementation Guidelines: Offering guidelines for implementing the framework in educational institutions, including training for evaluators and instructors.

Overall, the framework is likely designed to enhance the quality of teaching and learning by providing a structured approach to evaluating and improving teaching effectiveness.

#### **2.4 Learning Management System User Guide: Self & Peer Assessment (University of Melbourne):**

The paper "Learning Management System User Guide: Self & Peer Assessment (University of Melbourne)" likely provides a comprehensive guide on how to use the self and peer assessment features within the University of Melbourne's Learning Management System (LMS). A typical summary might include the following key points:

1. Introduction to Self & Peer Assessment: The guide starts with an overview of the importance and benefits of self and peer assessment in higher education, highlighting its role in promoting deeper learning, critical thinking, and feedback skills among students.
2. Accessing the Self & Peer Assessment Tool: It explains how students and instructors can access the self and peer assessment tool within the University of Melbourne's LMS. This includes step-by-step instructions on navigating to the tool and accessing assessment activities.
3. Creating Assessment Activities: The guide provides instructions on how instructors can create self and peer assessment activities within the LMS. This includes setting assessment criteria, deadlines, and instructions for students.

4. **Participating in Self & Peer Assessment:** For students, the guide explains how to participate in self and peer assessment activities. This includes instructions on how to submit their own work for assessment, how to evaluate peer submissions, and how to provide constructive feedback.
5. **Viewing Assessment Results:** Both students and instructors may be provided guidance on how to view assessment results within the LMS. This could include accessing feedback from peers, viewing grades, and reviewing assessment reports.
6. **Best Practices and Tips:** The guide may also include best practices and tips for both students and instructors to make the most of the self and peer assessment process. This could include advice on giving and receiving feedback, maintaining academic integrity, and using assessment results for learning improvement.
7. **Troubleshooting and Support:** Lastly, the guide provides information on troubleshooting common issues that may arise during the self and peer assessment process. It also includes information on where to seek further support, such as contacting IT support or academic staff.

Overall, the guide is designed to provide comprehensive support to both students and instructors using the self and peer assessment features within the University of Melbourne's LMS, aiming to enhance the effectiveness of assessment practices and promote student learning outcomes.

## **2.5 A Graph Analysis Method to Improve Peer Grading Accuracy for Blended Teaching Courses (Xing Du, Xingya Wang, and Yan Ma):**

The paper by Xing Du, Xingya Wang, and Yan Ma presents a comprehensive method for enhancing the accuracy of peer grading in blended teaching courses, focusing on the challenges of bias, inconsistency, and lack of expertise among peers. The authors begin by highlighting the importance of peer grading in large-scale courses for providing timely and personalized feedback to students, as well as promoting a deeper understanding of course material through the evaluation of peers' work.

To address these challenges, the authors propose a novel approach based on graph analysis, which they argue can effectively model the complex relationships between students and their grading behaviors. The method involves several key steps, starting with data preprocessing to clean and standardize the grading data. This includes identifying and correcting errors in grading, as well as normalizing scores to account for differences in grading standards among peers.

Next, the authors describe the process of constructing the grading graph, which involves representing students as nodes and grading relationships as edges. The graph is constructed based on various factors, including the similarity of grading patterns between students and the reliability of graders. This allows for the identification of clusters of students with similar grading behaviors, as well as the detection of outliers who may exhibit biased or inconsistent grading.

The paper also discusses the use of graph analysis techniques to analyze the grading graph and improve grading accuracy. This includes identifying and removing biased or unreliable graders, as well as adjusting grades based on the consensus of multiple graders. The authors demonstrate the effectiveness of their approach through experimental results, which show a significant improvement in grading accuracy compared to traditional peer grading methods.

Overall, the paper provides a detailed and innovative approach to improving peer grading accuracy in blended teaching courses. By leveraging graph analysis techniques, the proposed method offers a promising solution to the challenges of bias, inconsistency, and lack of expertise in peer grading, ultimately enhancing the quality of feedback and learning outcomes for students.

## **2.6 Fostering Peer Learning With a Game-Theoretical Approach in a Blended Learning Environment Seyede Fatemeh Noorani , Mohammad Hossein Manshaei , Mohammad Ali Montazeri, and Behnaz Omoomi:**

"Fostering Peer Learning With a Game-Theoretical Approach in a Blended Learning Environment" by Seyede Fatemeh Noorani, Mohammad Hossein Manshaei,



Mohammad Ali Montazeri, and Behnaz Omoomi explores the use of a game-theoretical approach to promote peer learning in a blended learning environment. The paper discusses the challenges of traditional learning methods and proposes a novel approach that combines game theory with blended learning techniques to enhance student engagement and collaboration.

The authors first highlight the limitations of traditional learning methods, such as passive learning and lack of student interaction, which can hinder the effectiveness of the learning process. They then introduce the concept of blended learning, which combines traditional face-to-face teaching with online learning activities, as a potential solution to these challenges.

The paper proposes a game-theoretical framework called "Peer-Incentive-Based Learning (PIBL)" to incentivize peer learning in a blended learning environment. The PIBL framework incorporates game elements, such as rewards and competitions, to motivate students to actively participate in learning activities and collaborate with their peers. The authors discuss how the PIBL framework can be implemented in practice, including the design of learning activities, assessment methods, and feedback mechanisms.

Furthermore, the paper presents a case study where the PIBL framework was applied in a real-world educational setting. The results of the case study demonstrate the effectiveness of the PIBL framework in promoting peer learning, improving student engagement, and enhancing learning outcomes.

Overall, the paper provides valuable insights into the potential of game-theoretical approaches to enhance peer learning in blended learning environments. The PIBL framework offers a novel and effective strategy to foster collaboration and engagement among students, ultimately improving the overall learning experience.

### **3. Summary and Shortcoming of above paper's:**

#### **1. LearnEval Peer Assessment Platform: Iterative Development Process and Evaluation:**

- **Summary:** The paper outlines the iterative development process of LearnEval, a peer assessment platform, emphasizing its features and evaluation methodologies. It aimed to simplify peer assessment while ensuring fairness, anonymity, and reliability in evaluations. Integration of machine learning algorithms was highlighted for enhancing its capabilities.
- **Evaluation:** The paper provides a comprehensive overview of the platform's development and evaluation, showcasing its positive impact on student learning outcomes and the assessment process. However, specific metrics and comparative studies with other platforms could enhance the evaluation's depth.

#### **2. Incentive Design in Peer Review: Rating and Repeated Endogenous Matching:**

- **Summary:** The paper proposes a novel incentive mechanism for peer review systems, combining rating and repeated endogenous matching to improve review quality and integrity. It addresses challenges in traditional peer review, such as biased reviews and lack of reviewer incentives.
- **Evaluation:** The paper presents a well-thought-out mechanism with theoretical analysis and simulations. However, real-world implementation and validation in academic publishing settings could provide more concrete evidence of its effectiveness.

#### **3. A Framework for Teaching Evaluation:**

- **Summary:** The paper proposes a framework for evaluating teaching effectiveness, covering objective setting, metrics selection, data collection, analysis, feedback mechanisms, continuous improvement, and implementation guidelines.
- **Evaluation:** The framework offers a structured approach to teaching evaluation but lacks specific details on metrics selection, data analysis methodologies, and implementation guidelines. Real-world application and validation would enhance its practicality.

**4. Learning Management System User Guide: Self & Peer Assessment:**

- **Summary:** The guide provides instructions on using self and peer assessment features within the University of Melbourne's Learning Management System, covering accessing the tool, creating assessment activities, participating in assessments, viewing results, best practices, and troubleshooting.
- **Evaluation:** The guide offers comprehensive support for users but could benefit from more visual aids and examples to enhance usability. It could also include case studies or user testimonials to illustrate its effectiveness.

**5. A Graph Analysis Method to Improve Peer Grading Accuracy for Blended Teaching Courses:**

- **Summary:** The paper proposes a graph analysis method to enhance peer grading accuracy in blended teaching courses, addressing challenges of bias, inconsistency, and lack of expertise among peers.
- **Evaluation:** The method presents a novel approach but lacks detailed discussion on practical implementation and scalability. Real-world validation and comparison with existing methods would strengthen its credibility.

**6. Fostering Peer Learning With a Game-Theoretical Approach in a Blended Learning Environment:**

- **Summary:** The paper introduces a game-theoretical framework, Peer-Incentive-Based Learning (PIBL), to promote peer learning in a blended learning environment, combining game elements with online learning activities.
- **Evaluation:** The framework offers a creative approach to enhance peer learning, but its effectiveness could be further validated through more extensive case studies and comparisons with traditional methods.

In conclusion, while these papers introduce innovative approaches and frameworks to enhance various aspects of education, including peer assessment, peer review, teaching evaluation, and peer learning, they could benefit from more robust evaluations,

practical implementations, and comparisons with existing methods to validate their effectiveness in real-world educational settings.

#### **4. Proposed Peer Evaluation System (Annexure-I for step-by-step implementation)**

The objective of this proposed system is to enhance the auto answer evaluation process in large classes and Massive Open Online Courses (MOOCs) by leveraging peer evaluation. This system aims to improve the efficiency, accuracy, and fairness of evaluations while incentivizing student participation and learning engagement.

##### **Key Features:**

##### **1. Criteria Alignment:**

- Students evaluate their peers' answers based on the same criteria used by the automated system.
- Criteria are clearly defined and provided to students to ensure consistency and fairness in evaluations.

##### **2. Peer Evaluation Groups:**

- Each answer sheet is evaluated by multiple peers to ensure diverse perspectives and reduce bias.
- Peers are assigned randomly to evaluate answers sheets, ensuring a fair distribution of evaluations.

##### **3. Quality Control Mechanisms:**

- To maintain the reliability of evaluations, a subset of answers sheets is selected for double-blind evaluation by both peers and TAs.
- A consensus mechanism is employed to resolve discrepancies between peer evaluations and ensure accuracy.

##### **4. User-Friendly Interface:**

- The system features a user-friendly interface that guides students through the evaluation process.
- It provides clear instructions, examples, and rubrics to help students understand the evaluation criteria.

5. Timely Feedback:

- The system ensures timely feedback to students by setting deadlines for evaluations and providing automated reminders.
- Feedback is provided anonymously to maintain confidentiality and encourage honest evaluations.

6. Scalability:

- The system is designed to be scalable, allowing it to handle a large number of evaluations simultaneously.
- It can be easily integrated into existing learning management systems (LMS) or MOOC platforms.

**Workflow:**

1. Assignment Submission:

- Students submit their answers to assignments or quizzes through the online platform.

2. Peer Evaluation Assignment:

- Peers are assigned a subset of answers sheets to evaluate based on the criteria provided.

3. Quality Control Evaluation:

- A subset of answers sheets is selected for double-blind evaluation by both peers and TAs.
- Discrepancies between peer evaluations and TA evaluations are resolved through a consensus mechanism.

4. Health Points Allocation:

- Peers are awarded health points based on the accuracy and consistency of their evaluations.
- Health points are assigned to the peer's and these points will help them to get additional marks in the whole course.

5. Feedback and Results:

- Students receive feedback on their submissions, including peer evaluations and TA feedback.

- They can use this feedback to improve their future submissions and enhance their learning experience.

The proposed peer evaluation system for auto answer evaluation offers a comprehensive framework to enhance the evaluation process in large classes and MOOCs. By leveraging peer evaluation, the system aims to improve the efficiency, accuracy, and fairness of evaluations while incentivizing student participation and engagement.

## **5. Methodology: (Annexure-II for Code and Implementation)**

### **5.1 Participants**

A cohort of 150 students enrolled in a specific course participated in the study. Among them, 132 students actively participated in the examination.

### **5.2 Examination Structure**

The examination consisted of 15 questions of various types, designed to comprehensively assess the students' understanding of the course material.

### **5.3 Answer Sheet Handling**

Following the examination, the answer sheets were scanned and stored in a Google Drive folder for further analysis and evaluation.

### **5.4 Anonymization and Distribution**

To ensure fairness and impartiality, each of the 132 answer sheet copies was assigned a unique identification number. A meticulously planned script was then utilized to distribute these copies among students and Teaching Assistants (TAs) in seven groups labeled as G0, G1, G2, ..., G7.

### **5.5 Evaluation Process**

Each question on every test copy was evaluated by a minimum of 3 students and one TA to ensure comprehensive evaluation. Additionally, a unique cross-evaluation system was implemented, whereby different groups were assigned to evaluate the test copies of another group to mitigate potential bias in the evaluation process. Detailed marking

schemes with solutions were provided to every student and TA to standardize the evaluation process and maintain consistency.

## **6. Results:**

Peer evaluation is a method for enhancing the accuracy and fairness of assessments in educational settings. In this study, we present a detailed analysis of a peer evaluation system implemented in a university course to evaluate student responses to examination questions. The system involved 132 students, each of whom had their responses evaluated by peers and Teaching Assistants (TAs) to ensure question-wise accuracy. Anonymity was maintained throughout the evaluation process, with each student's entry linked to a unique ID.

The peer evaluation system was implemented as follows:

1. Anonymization: Each student's entry number was linked to a unique ID to ensure anonymity throughout the evaluation process.
2. Peer Evaluation: Every question on every test copy was evaluated by a minimum of three students and one TA to ensure fairness and accuracy.
3. Cross-Evaluation: Different groups were assigned to evaluate the test copies of another group to mitigate potential bias in evaluation.
4. Detailed Marking Scheme: A detailed marking scheme with solutions was provided to every student and TA to standardize the evaluation process.

The results of the analysis indicate that the peer evaluation system was highly effective in ensuring question-wise accuracy. By comparing the evaluations done by peers and TAs, we were able to identify areas where the auto answer evaluation system may need improvement. The anonymization process also helped in maintaining the integrity of the evaluation process.

| Uniqu<br>e id | Evaluated<br>by | Q1 | Q2 | Q3 |    | Q4 | Q5 | Q6 |    | Q7 | Q8 | Q9 |   | Q10 | Q11 | Q12 |   | Q13 | Q14 | Q15 |   | Grand<br>Total |
|---------------|-----------------|----|----|----|----|----|----|----|----|----|----|----|---|-----|-----|-----|---|-----|-----|-----|---|----------------|
| 487           | Peer 1          | 0  | 0  | 3  | 3  | 0  | 10 | 0  | 10 | 0  | 0  | 5  | 5 | 0   | 0   | 2   | 2 | 0   | 0   | 3   | 3 | 23             |
|               | Peer 2          | 0  | 0  | 3  | 3  | 0  | 10 | 0  | 10 | 0  | 0  | 8  | 8 | 0   | 0   | 0   | 0 | 0   | 2   | 3   | 5 | 26             |
|               | Peer 3          | 0  | 0  | 3  | 3  | 0  | 10 | 0  | 10 | 0  | 0  | 5  | 5 | 0   | 4   | 0   | 4 | 0   | 2   | 3   | 5 | 27             |
|               | TA              | 0  | 0  | 10 | 10 | 0  | 10 | 0  | 10 | 0  | 0  | 5  | 5 | 0   | 0   | 0   | 0 | 2   | 0   | 3   | 5 | 30             |

Table-I

Evaluation process is done in such a way that different groups are evaluating answer scripts of another group as shown in Table-II

| Name of<br>Peer    | link to drive              | Test Script<br>for<br>Checking | Group for<br>which<br>checking is<br>done | Q NO | Q NO | Q NO | link to<br>google<br>sheet | SHEET<br>NAME |
|--------------------|----------------------------|--------------------------------|---|------|------|------|----------------------------|---------------|
| Vrushank<br>Ahire  | Please refer<br>Annexure-1 | G7                             | G1  | Q1   | Q2   | Q3   | Please refer<br>Annexure-1 | T5            |
| Aryan<br>Verma     | Please refer<br>Annexure-1 | G7                             | G1  | Q4   | Q5   | Q6   | Please refer<br>Annexure-1 | T6            |
| Manish<br>Kumar    | Please refer<br>Annexure-1 | G7                             | G1  | Q7   | Q8   | Q9   | Please refer<br>Annexure-1 | T9            |
| Pratik<br>Kumar    | Please refer<br>Annexure-1 | G7                             | G1  | Q10  | Q11  | Q12  | Please refer<br>Annexure-1 | T10           |
| T.Shayan           | Please refer<br>Annexure-1 | G7                             | G1  | Q13  | Q14  | Q15  | Please refer<br>Annexure-1 | T11           |
| Ajmeera<br>Vasuram | Please refer<br>Annexure-1 | G7                             | G1  | Q1   | Q2   | Q3   | Please refer<br>Annexure-1 | T13           |
| Arav<br>Bhargava   | Please refer<br>Annexure-1 | G7                             | G1  | Q4   | Q5   | Q6   | Please refer<br>Annexure-1 | T14           |
| Prashant<br>Kumar  | Please refer<br>Annexure-1 | G7                             | G1  | Q7   | Q8   | Q9   | Please refer<br>Annexure-1 | T16           |



|                  |                         |    |    |     |     |     |                         |     |
|------------------|-------------------------|----|----|-----|-----|-----|-------------------------|-----|
| Animan Naskar    | Please refer Annexure-1 | G7 | G1 | Q10 | Q11 | Q12 | Please refer Annexure-1 | T17 |
| Hartik Arora     | Please refer Annexure-1 | G7 | G1 | Q13 | Q14 | Q15 | Please refer Annexure-1 | T18 |
| Jyoti            | Please refer Annexure-1 | G7 | G1 | Q1  | Q2  | Q3  | Please refer Annexure-1 | T19 |
| Pranav Menon     | Please refer Annexure-1 | G7 | G1 | Q4  | Q5  | Q6  | Please refer Annexure-1 | T20 |
| Jitender         | Please refer Annexure-1 | G7 | G1 | Q7  | Q8  | Q9  | Please refer Annexure-1 | T25 |
| Vanjivaka Sairam | Please refer Annexure-1 | G7 | G1 | Q10 | Q11 | Q12 | Please refer Annexure-1 | T26 |
| Aadit Mahajan    | Please refer Annexure-1 | G7 | G1 | Q13 | Q14 | Q15 | Please refer Annexure-1 | T28 |

Complete data of 132 students appeared in the exam with Peer average and SD:

[illegible]



|    |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |
|----|-----------------|--|------------------------------|----------------------|------------|------------|-----|----|----------------|---|-----|----|--------|
| D  | 304037565545927 | 5887995915542809                             | 1086305941940900871501447287 | 10380037309576922874 | 211        | 0357646258 |     |    |                |   |     |    |        |
| e  | 430985356515051 | 77932791702022636453598006448280930919368757 | 53610482924477839587         | 114                  | 8963603247 |            |     |    |                |   |     |    |        |
| vi | 688415954243574 | 48254637231668434                            | 8114167                      | 5849544              | 6          | 5214       | 477 | 92 | 37885275369898 | 4 | 257 | 37 | 256771 |
| a  |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |
| t  |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |
| i  |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |
| o  |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |
| n  |                 |  |                              |                      |            |            |     |    |                |   |     |    |        |

Table-III

Standard Deviation

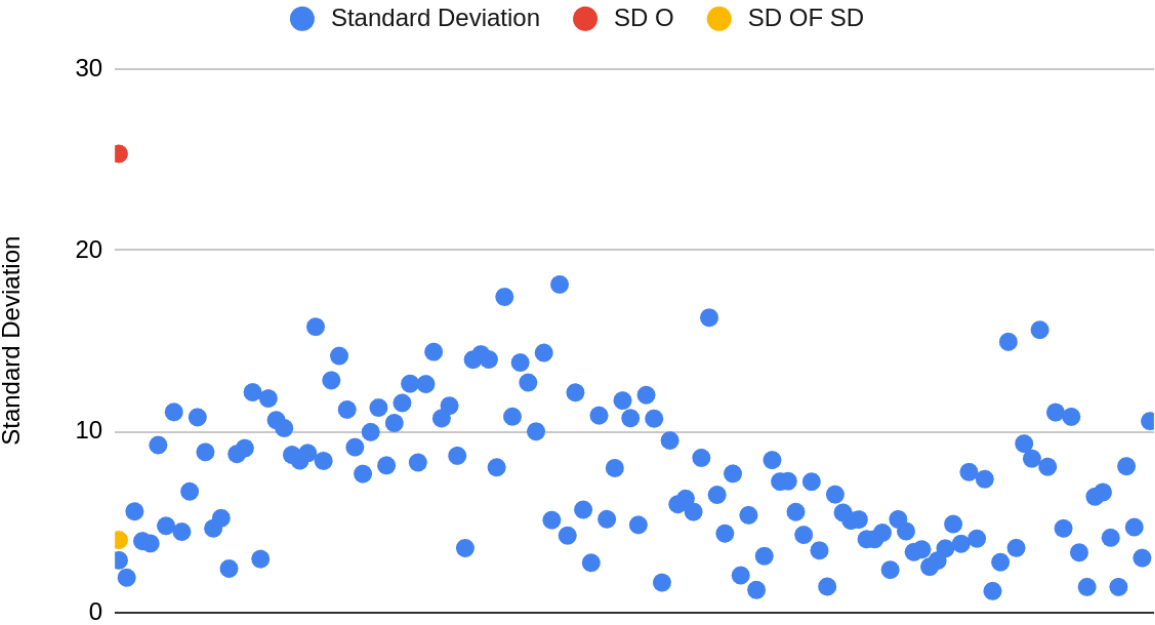


Figure-1

Plots showing complete evaluation data and it's analysis (as per data obtained in Table-III)

## **7. Conclusion:**

The proposed peer evaluation system represents a significant advancement in addressing the challenges associated with manual evaluation in large classes and Massive Open Online Courses (MOOCs). By harnessing the collective intelligence of students, this system offers a viable solution to distribute the evaluation workload, alleviate the burden on instructors, and expedite feedback delivery to students. Furthermore, it has the potential to enhance student engagement, foster critical thinking skills, and promote a collaborative learning environment.

The effectiveness and scalability of this system make it a promising avenue for further research and implementation in educational settings. Future studies could focus on evaluating the impact of the peer evaluation system on student learning outcomes, instructor workload, and overall course satisfaction. Additionally, exploring ways to optimize the system's design and integration within existing educational technology frameworks could further enhance its utility and adoption.

In conclusion, the proposed peer evaluation system represents a valuable contribution to the field of education technology. Its innovative approach to evaluation not only addresses current challenges but also opens up new possibilities for enhancing the learning experience in large classes and MOOCs. By continuing to refine and explore this system, educators can potentially transform the way assessments are conducted and create more engaging and effective learning environments for students.

## **8. Future Work:**

1. **Developing Scalable Software:** We will develop a scalable software with a more user-friendly interface for the peer evaluation process. The software allows students to easily access and evaluate their peers' answer sheets, will also provide prompts for feedback, and calculates scores based on a predefined criteria.

## 9. References:

1. *LearnEval Peer Assessment Platform: Iterative Development Process and Evaluation* (Gabriel Badea and Elvira Popescu, Senior Member, IEEE).
2. *Incentive Design in Peer Review: Rating and Repeated Endogenous Matching* (Yuanzhang Xiao, Florian Dorfler, and Mihaela van der Schaar, Members/Fellows, IEEE)
3. *A Framework for Teaching Evaluation* (Divya Nalla, Department of Computer Science, Nalla Malla Reddy Engineering College).
4. *Learning Management System User Guide: Self & Peer Assessment* (University of Melbourne).
5. *A Graph Analysis Method to Improve Peer Grading Accuracy for Blended Teaching Courses* (Xing Du, Xingya Wang, and Yan Ma).
6. *Fostering Peer Learning With a Game-Theoretical Approach in a Blended Learning Environment* (Seyede Fatemeh Noorani , Mohammad Hossein Manshaei , Mohammad Ali Montazeri, and Behnaz Omoomi)

## **Annexure-I (Step-by-Step process of running Peer Evaluation)**

### **INTRODUCTION**

The learning management system (LMS) like Moodle, Canvas etc has been used only for assignment checking.

We have designed an Automated Peer Evaluation System with the aim to save crucial and important work-hours of the teachers and TAs which can be used in more productive outcomes to improve the education delivered to the students.

This will also enhance the fairness system amongst the students and students will get to know about the feedback of their performance.

The students will also get a hang of the deeper level of the subject while evaluating others. The best way to learn a subject is by teaching it.

Thus the peer evaluation will work in multiple ways to improve the education outcomes of the course.

### **The System**

The System has four major components, Question Paper Printing which includes custom printing of the Unique ID on the sheets to ensure that the students get the sheets randomly. The next part is the upload and renaming of the answer sheets using the OCR techniques. The next part is the Pre-Evaluation steps which sets up the peer evaluation groups and sends the details to each student for their specific evaluation on their portal. The portal contains the link to the drive folder in view-only mode and a link to the evaluation spreadsheet. The next and last part is Post-Evaluation steps which does the consolidation of the marks and then sends the individual marks to the students.

### **Components**

The system has four major components, including the Printing Manager, the Upload and rename files, the Pre-Evaluation steps, and the Post-Evaluation steps encompassing the entire testing process. The making of the test paper, collecting them and uploading and renaming them and then evaluating the answer scripts, and then consolidating the results.

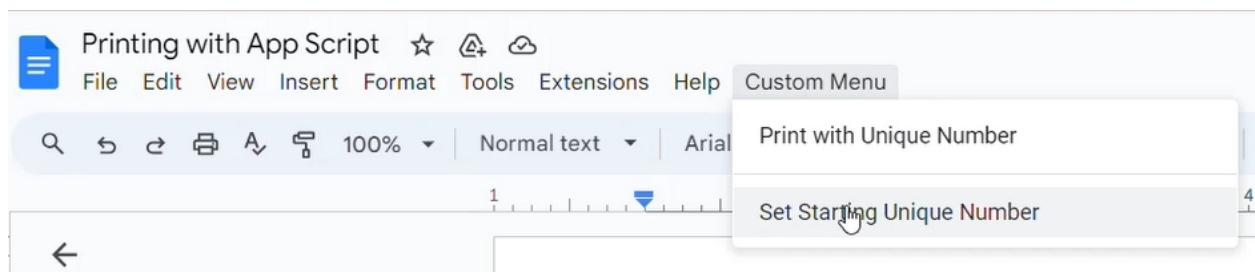
**The Peer Evaluation System Portal components are intuitive and easy to use. The user will be able to understand the components easily.**

# QUESTION PAPER PRINTING

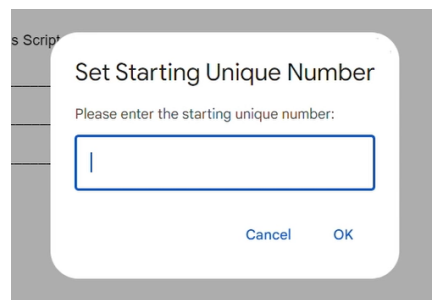
## Introduction and Usage

The Question Paper Printing system is hosted on Google Docs and uses add-ons to make the printing process personalized and easy to print the sheets with a special unique ids. These unique id's get mapped to the students once they have given the test/quiz and will help in tracking the candidate throughout the entire evaluation process.

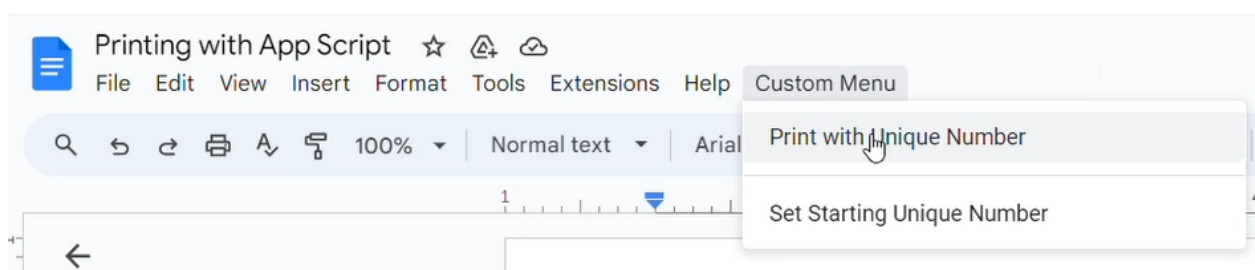
To start, we go to the Google Doc and make the question paper, giving the space required for writing the answers to the questions. Then we go to the **Custom Menu** → **Set Starting Unique Number**



This will open the dialogue box to enter the starting unique number for each student. This will increment automatically and only the first number is required to be entered.



Next, we need to go to the **Custom Menu** → **Print with Unique Number** to print the question paper sheets. These will be done in batches and each batch will have a different unique ID corresponding to the candidate.



# PEER-EVALUATION SYSTEM

## Introduction

The Pre-Evaluation steps should be run after the test has been taken and the scanned files are uploaded through the portal and the renaming of files is done and the files are in the folder in Google Drive. These scanned files are the answer scripts of the candidates.

## Guide to Use

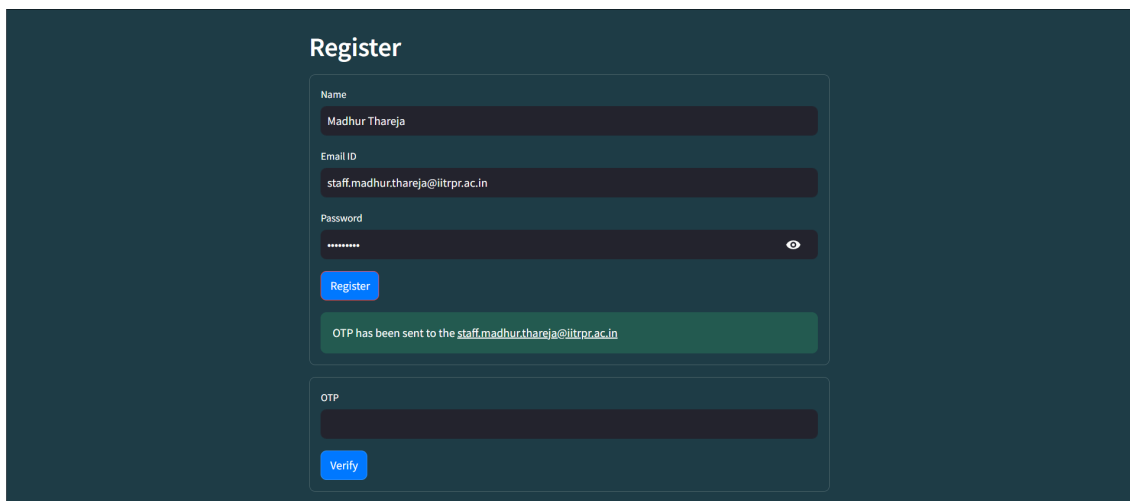
- **Registration & Login window: -**

For the registration window we have three fields that are Name, Email Id and Password. Once the fields are filled and we press the Register button, if the email id is valid the user will get the OTP through the email id and they can enter the one time password in the OTP field to verify their registration.

**Note: - The password must be 8 characters long with a capital letter, small letter, a numerical digit and a special character out of the following characters “ @\$!%\*?& ”.**

The below are the steps to do the new user registration: -

Step 1: - Fill the details and press the “Register” button.

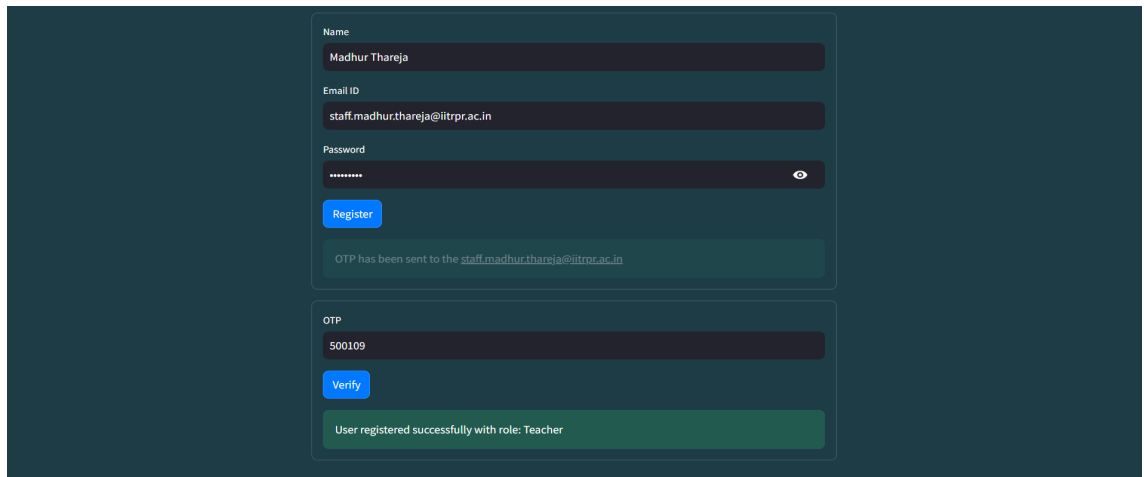


The screenshot shows a web form titled "Register" on a dark blue background. The form has three input fields: "Name" with the value "Madhur Thareja", "Email ID" with the value "staff.madhur.thareja@iitrpr.ac.in", and "Password" with masked characters "\*\*\*\*\*". A blue "Register" button is positioned below the password field. Directly beneath the button, a green notification box displays the message "OTP has been sent to the staff.madhur.thareja@iitrpr.ac.in". Below this notification, there is an "OTP" input field containing the value "000000" and a blue "Verify" button.

In the above window the OTP section of the form will be visible only if the OTP is sent successfully and then we can fill the OTP.



Step 2: - Check the email for the one time password (Spam also) and fill the one time password in the OTP field and press the “Verify” button.

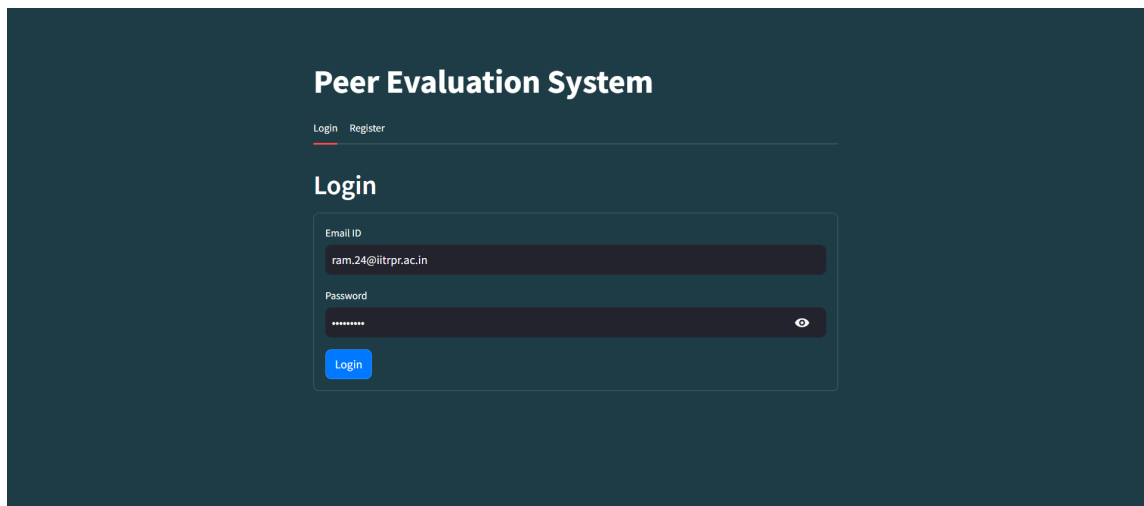


The image shows a registration and OTP verification interface. The top section is for registration, with fields for Name (Madhur Thareja), Email ID (staff.madhur.thareja@iitrpr.ac.in), and Password (masked with dots). A blue 'Register' button is below the password field. A green message bar states: 'OTP has been sent to the staff.madhur.thareja@iitrpr.ac.in'. The bottom section is for OTP verification, with a field for the OTP (500109) and a blue 'Verify' button. A green message bar at the bottom states: 'User registered successfully with role: Teacher'.

After the above successful registration we can navigate to the Login page and can do the Login and we will get the dashboard as per the role which is assigned based on the email id.

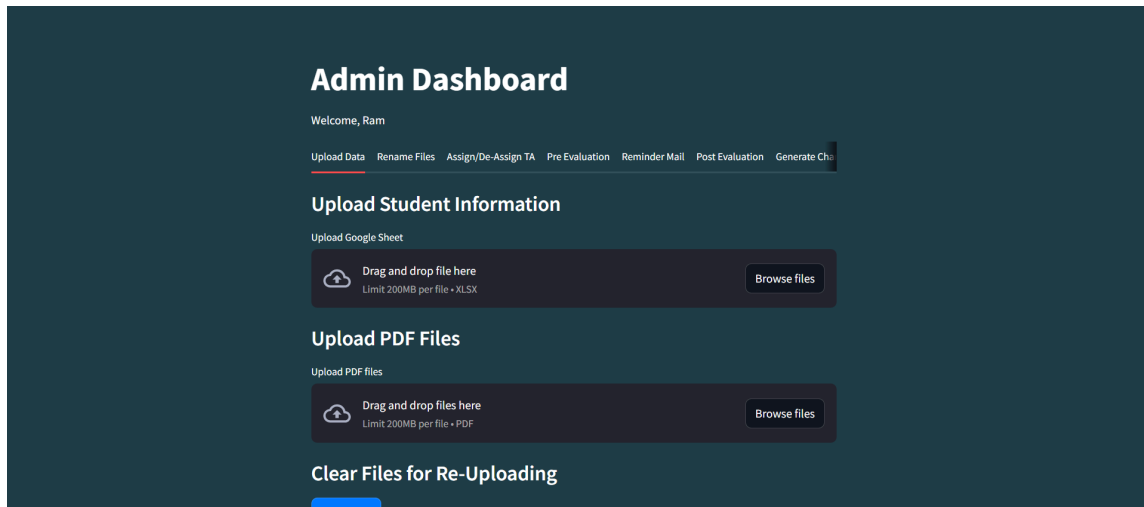
Step 3: - Logging in to the portal.

Enter the details on to the portal as shown:



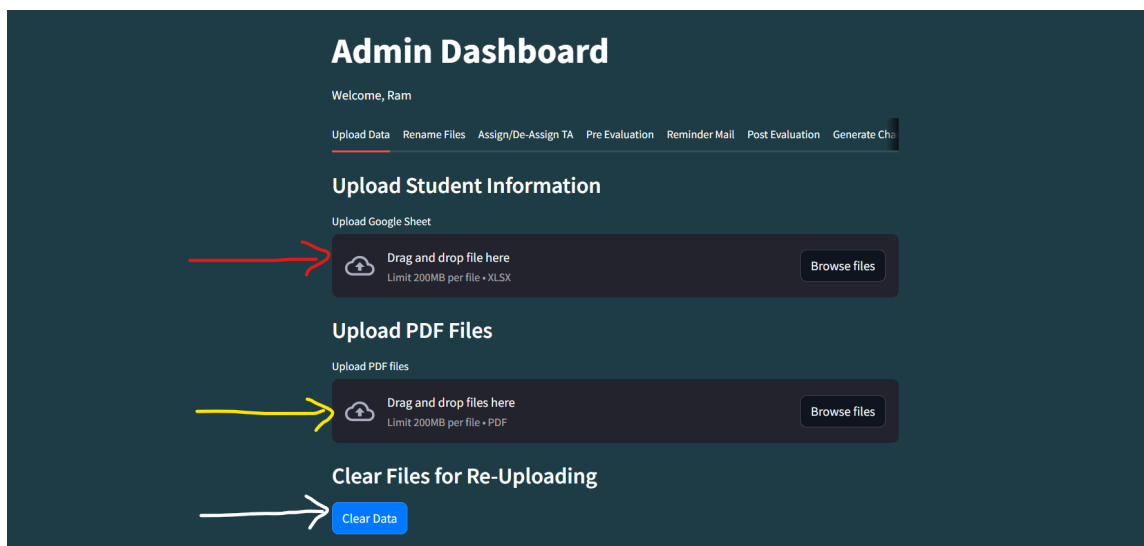
The image shows the login interface for the 'Peer Evaluation System'. At the top, there are links for 'Login' and 'Register', with 'Login' being the active link. Below the links is the 'Login' heading. The login form has two fields: 'Email ID' (ram.24@iitrpr.ac.in) and 'Password' (masked with dots). A blue 'Login' button is at the bottom of the form.

After the successful login we will be able to see the dashboard as per the role, here the Admin dashboard will be visible to us as we have logged in as the Admin.



- **File Upload window: -**

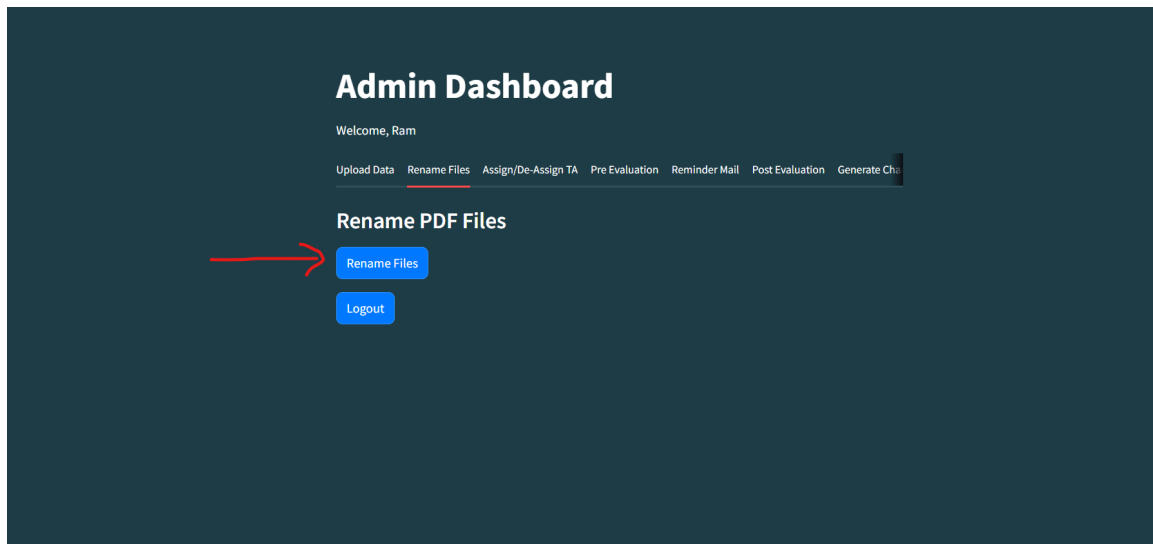
1. Use the “Upload Student Information” uploader depicted with red arrow, for uploading the sheet consisting of the mapped email id with the unique id.
2. Use the “Upload PDF Files” uploader depicted with yellow arrow, for uploading the scanned answer sheets.
3. Use the “Clear Data” button depicted with white arrow for deleting all files in case of an error or problem.



- **Rename Files window: -**

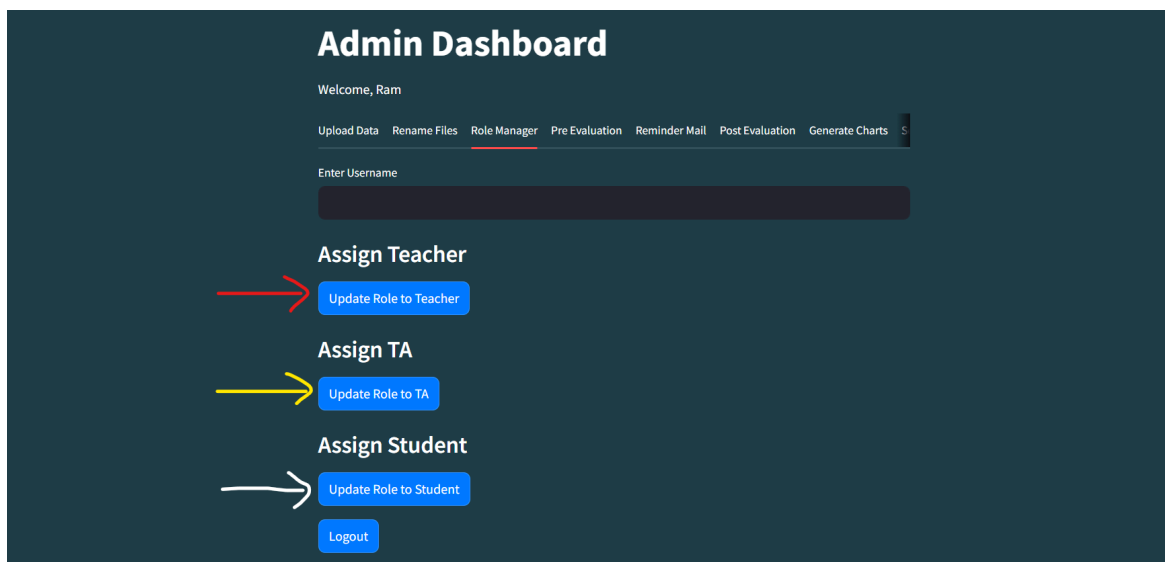
After uploading the files successfully, now we are ready to rename the uploaded files ( can be done using the “Rename Files” button depicted with red arrow) according to the

assigned unique id which is present in the top left corner of the pdf files which we are extracting using the OCR technique.



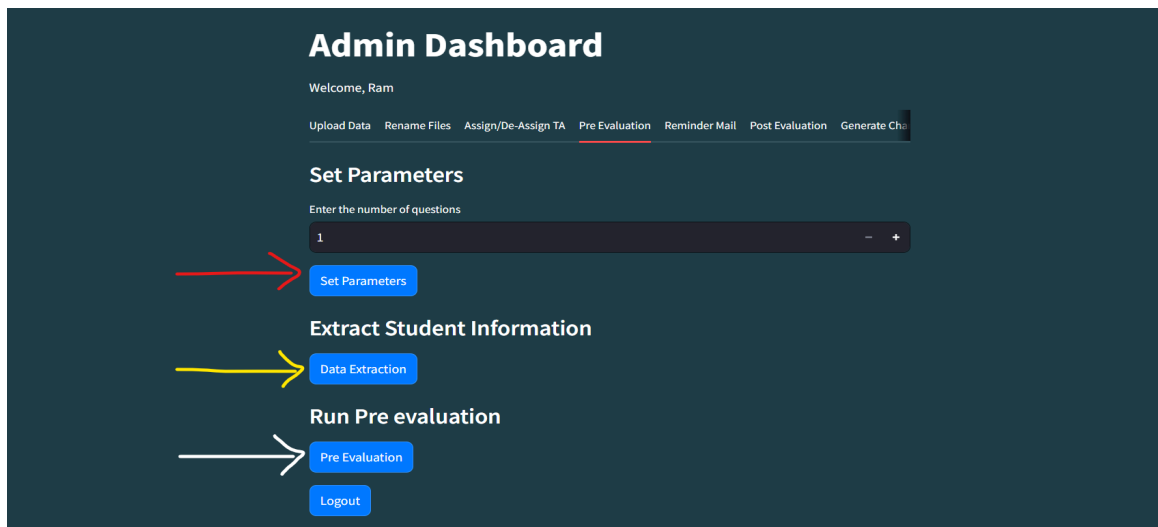
- **Role Manager window: -**

If Admin or a teacher wants to assign a user a different role than they can use the below window. After entering the username the teacher can use the button as per the requirement and can assign a student to/from the TA duty. Also if we want to make any Teacher a student due to any reason we can do it from the Admin's I'd. The button depicted with the red arrow helps to assign the "Teacher" role and the button depicted with the yellow arrow is to assign the TA role. The button depicted with the white arrow is for assigning the user a Student role.



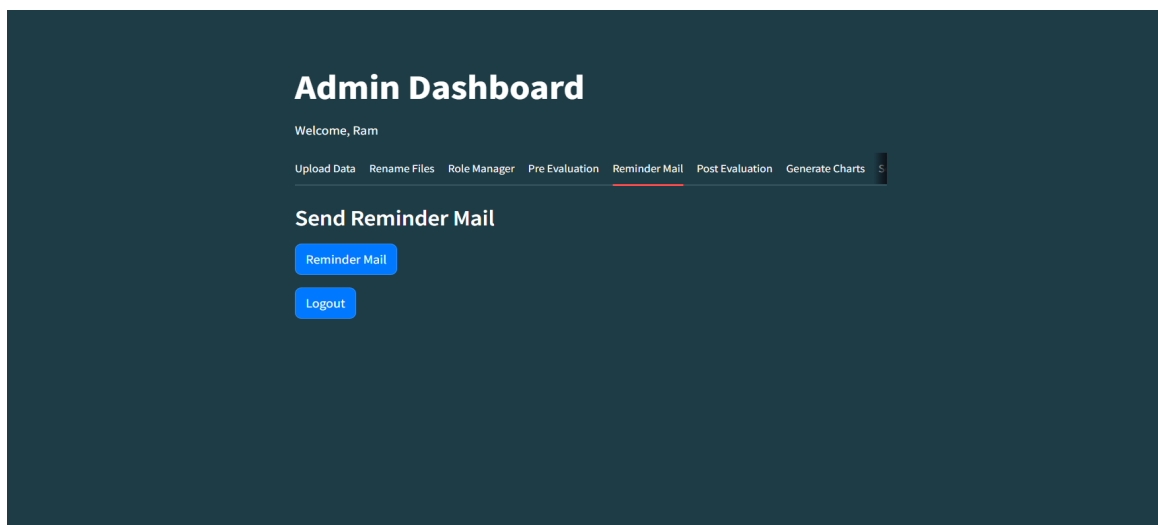
- **Pre Evaluation window: -**

The Set Parameter section which has the field for the number of questions and a “Set Parameters” button (depicted with red arrow) will help the user to set the number of questions that are there in the current quiz/test. The “Data Extraction” button (depicted with yellow arrow) will extract all the information of the students from the uploaded sheet to the main processing sheet and will clear the uploaded sheet from the folder to optimize the space. The “Pre Evaluation” button (depicted with white arrow) will assign all the data to the student to evaluate and they will be able to see the sheets that they are going to evaluate.



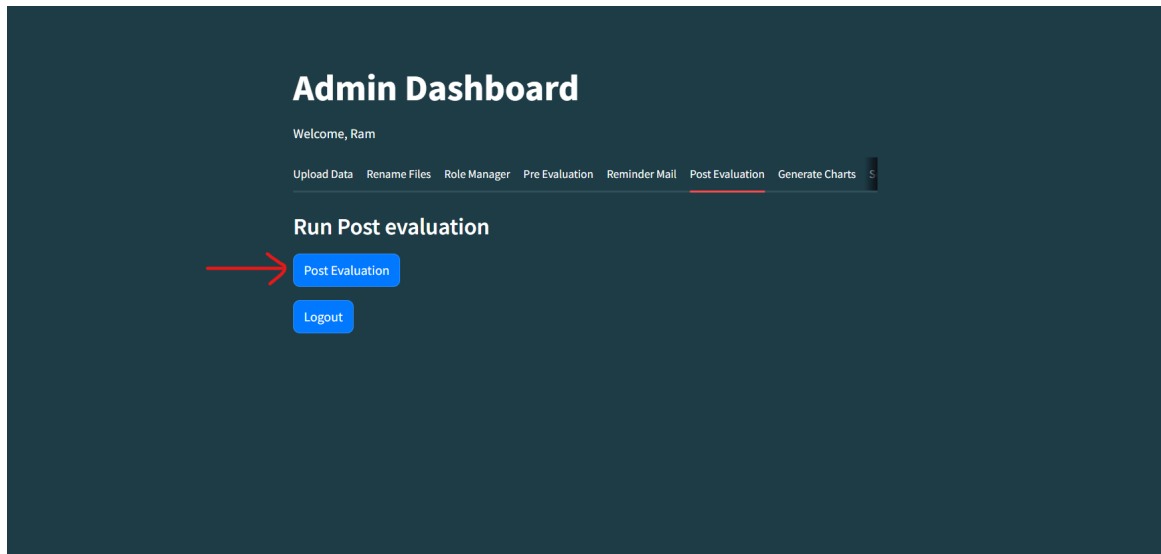
- **Reminder Mail window: -**

The “Reminder Mail” button will help the Teacher and TA to send a reminder to all those students who haven’t done the evaluation till now.



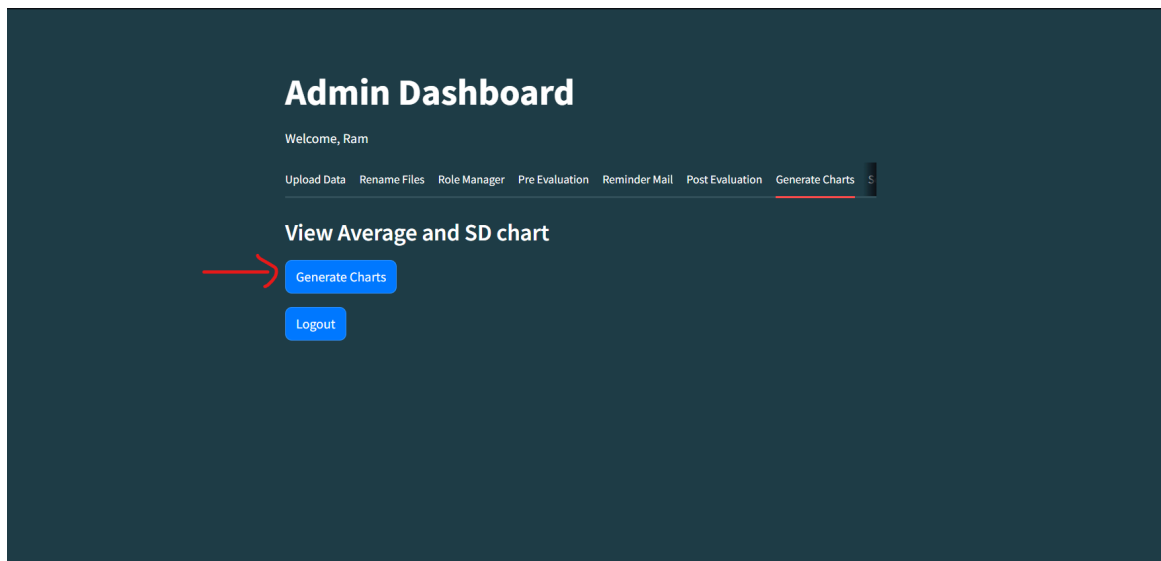
- **Post Evaluation window: -**

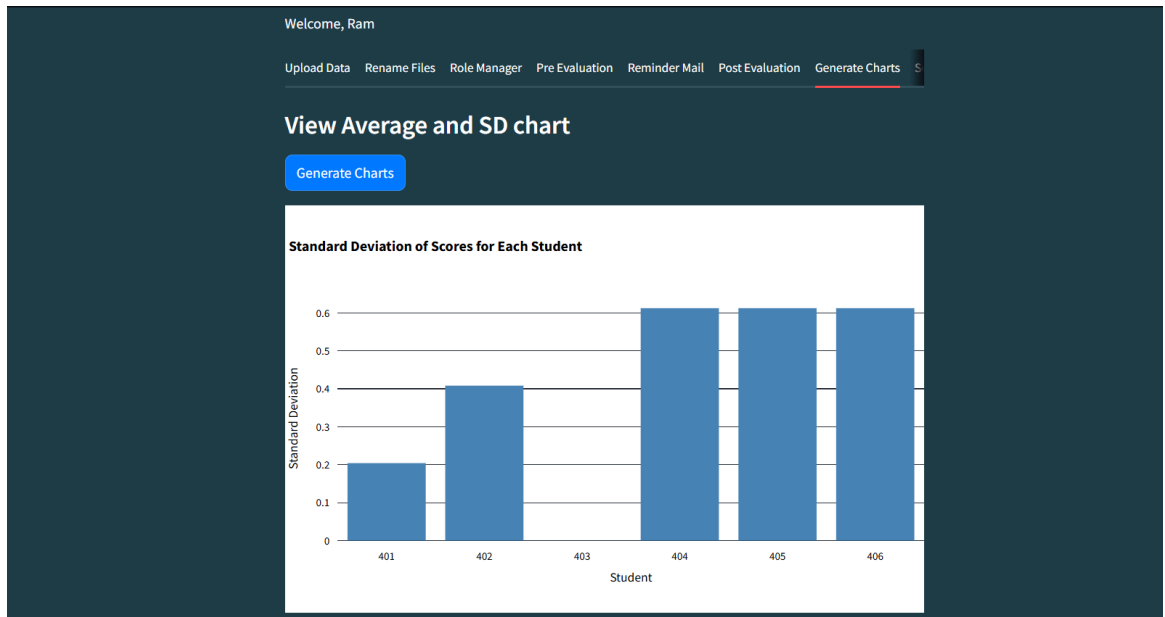
The “Post Evaluation” button depicted with a red arrow helps Teachers and TA’s to consolidate the marks of the student at one place and this helps further in releasing the marks.



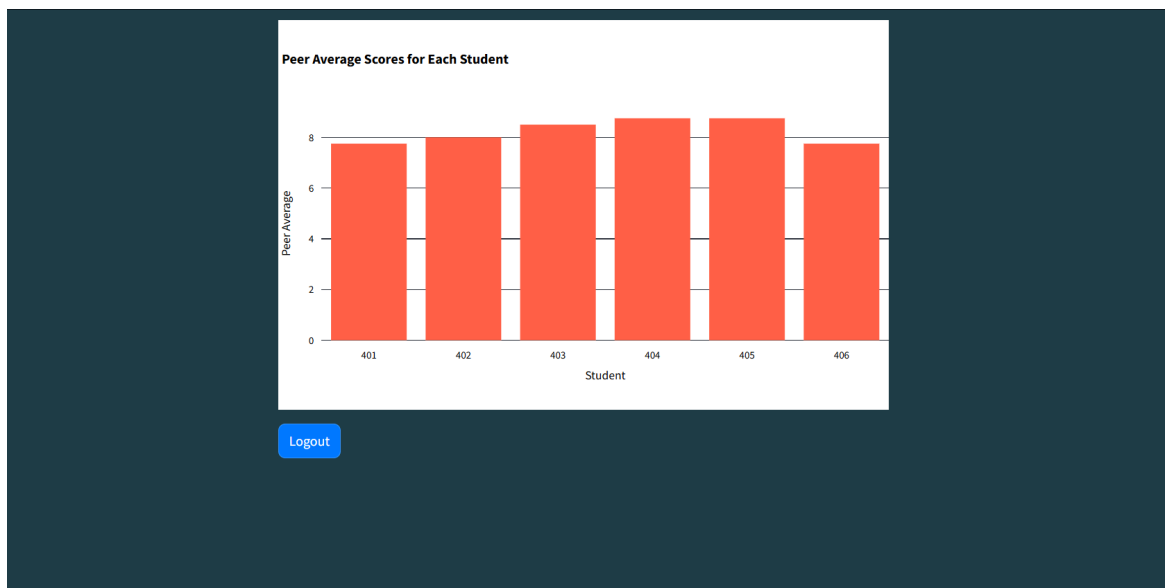
- **Generate Charts window: -**

The “Generate Charts” button depicted with a red arrow helps Teachers and TA’s to visualise the plots of the average marks and standard deviation of the students.





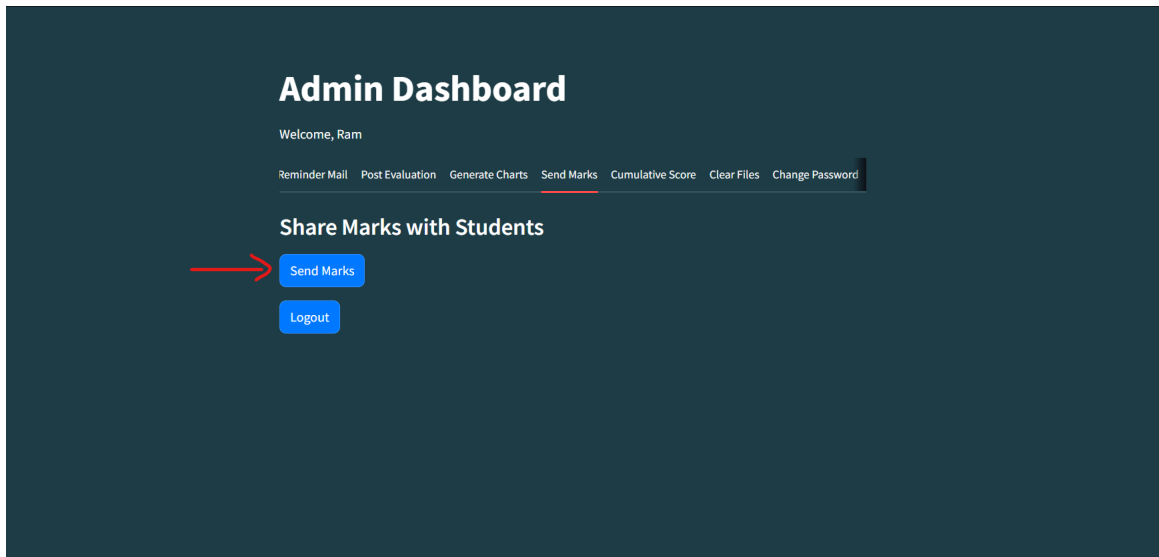
The above plot shows the standard deviation of each student. We can use the above plot for visualizing the standard deviation of average marks among all students.



The above plot shows the average marks of each student. We can use the above plot for visualizing the average marks of each student and we can see how the students are performing in the class.

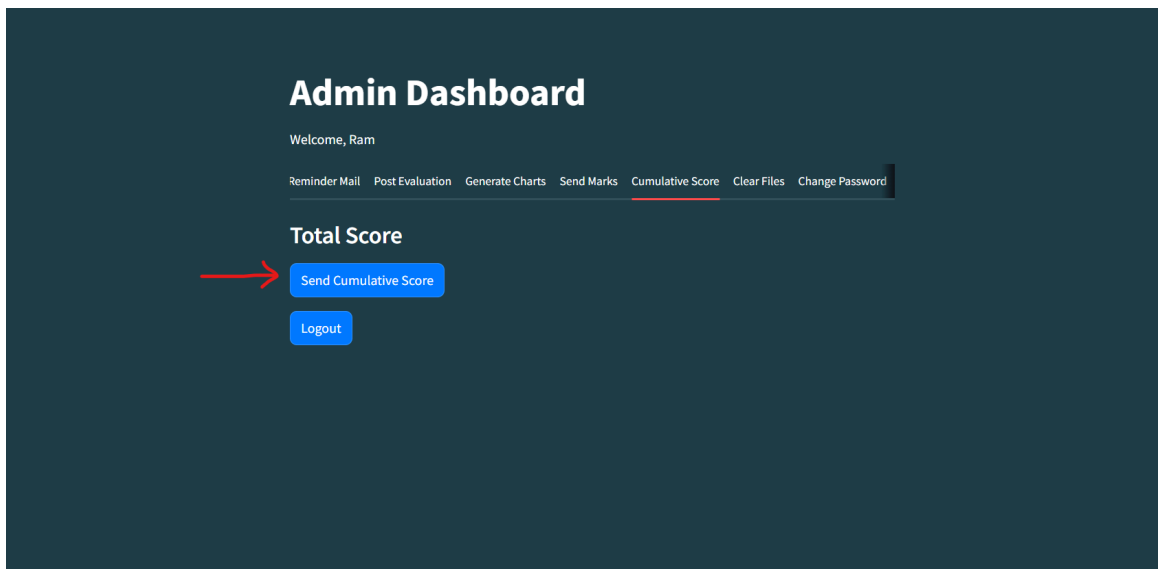
- **Send Marks window: -**

The “Send Marks” button depicted with a red arrow helps Teachers and TA's to release the Quiz/Test marks to the students.



- **Cumulative Score window: -**

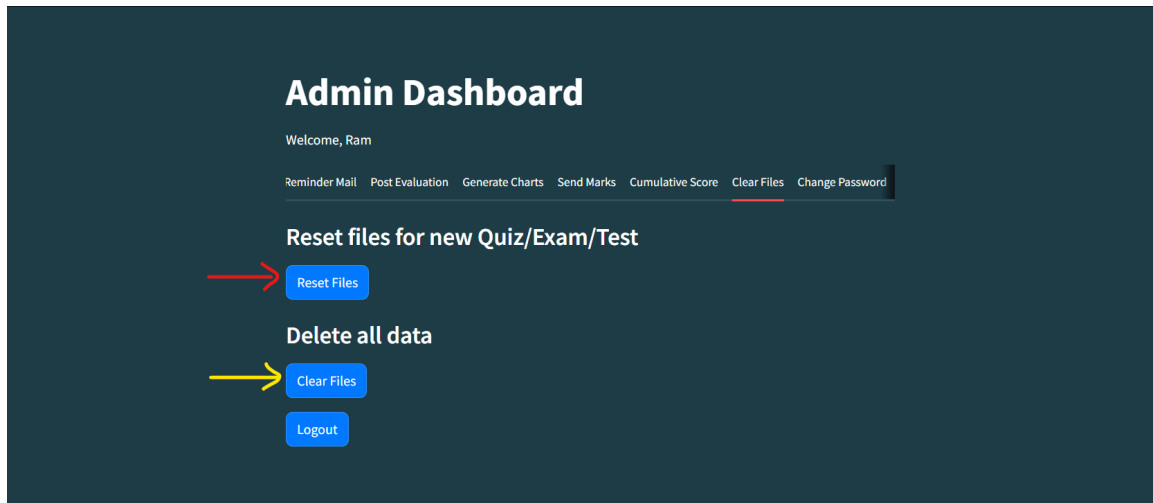
The “Send Cumulative Score” button depicted with a red arrow will help the Teachers and TA's to send the total marks of the students scored in the subject. (We assume that the scaling is done explicitly).



- **Clear Files window: -**

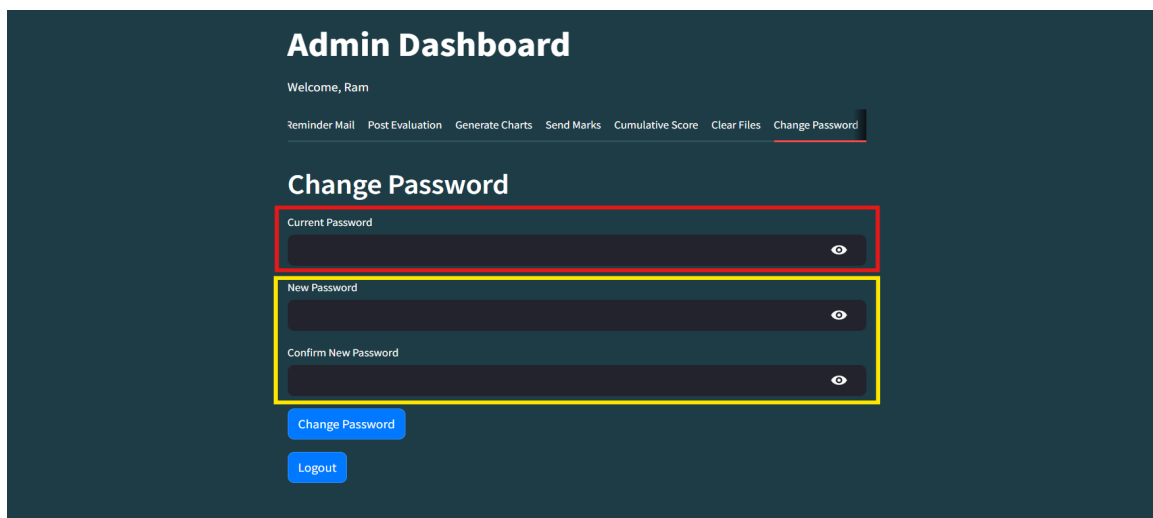
The “Reset Files” button depicted with a red arrow will help the Teachers and TA’s to clear the files/backend data. This is provided as a precautionary measure and the script is already handling the clearing of the files.

The “Clear Files” button depicted with a yellow arrow will help the Teachers and TA’s to clear all data including the sheets which is serving as the backend for our system. It is serving as the reboot to our evaluation system.



- **Change Password window: -**

The “Change Password” functionality has three input fields, the first field depicted with red box takes the current password of the user and the other two fields depicted with yellow box are taking the new password and confirmation for the new password, if both matches the password get changed in the server.

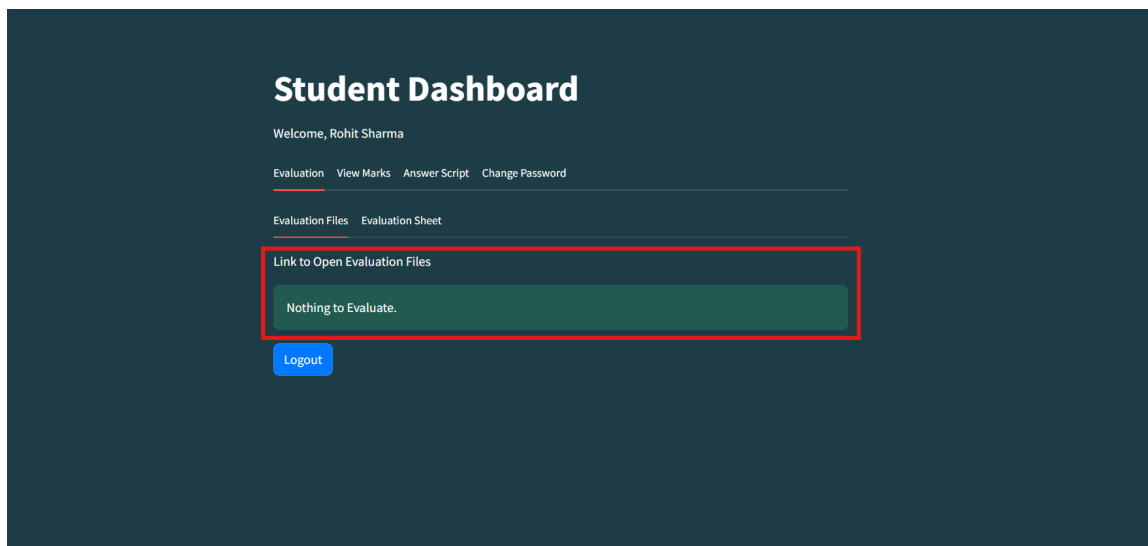




The above whole step by step guide is similar for the Admin, Teacher and TA dashboard. For depiction the Admin dashboard is used because it has all the functionalities from all the three dashboards. The other two dashboards have limited functionality according to the roles. For example the Upload file option is not provided to the teacher because the TA's will do that for them and the functionality like Role Manager is not given to the TA's because we want that only Teacher or Admin can do changes to the roles of the user's. The Admin has the access to make somebody as the Teacher (if we want to make somebody as a teacher who is not entitled to be the teacher for the course), because this is a very crucial role we are not giving it's control to the Teacher's even.

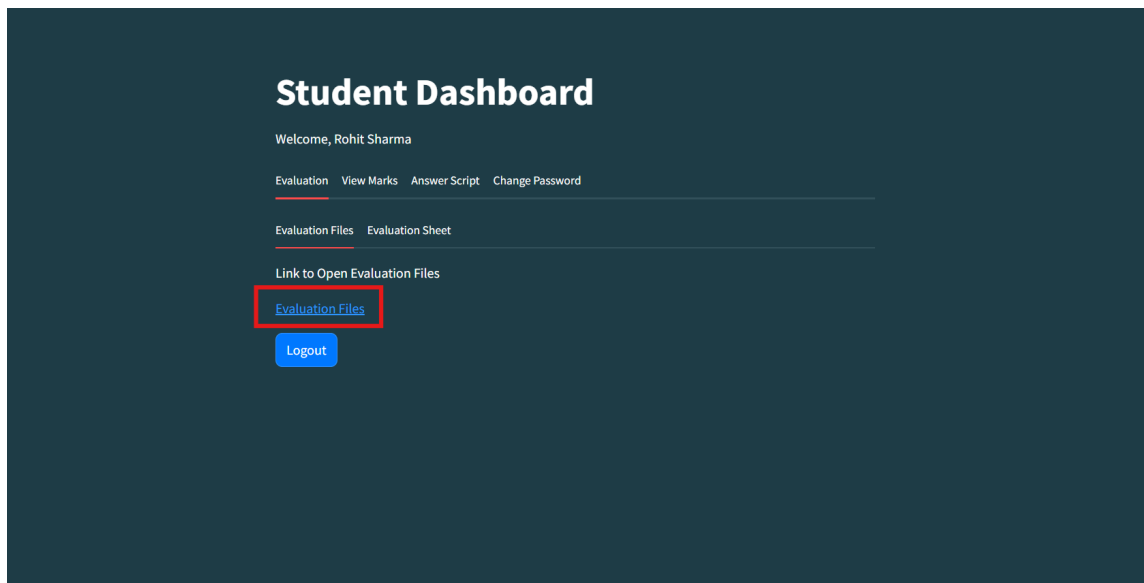
- **Student dashboard ( Evaluation) window: -**

The student dashboard will give the below message if there is nothing to evaluate for a student. The message for the “Evaluation Files” window is depicted with the red box for clarity. Also, the message for the “Evaluation Sheet” window would be the same.

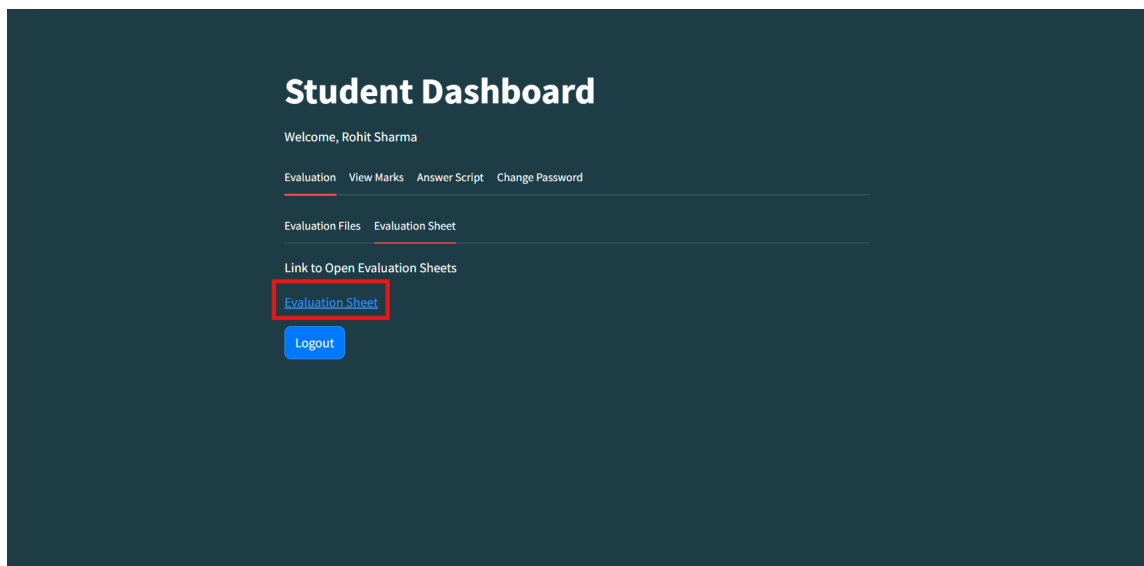


If there is something to evaluate for the student then they will get the link under the “Evaluation Files” section and when they click the link, a google drive folder will appear in a new tab of the browser and they will be able to see all the files they need to evaluate. Also, for the “Evaluation Sheet” a link will appear and they will be able to open a sheet where they can update the marks of the assigned students or peer's.

The sample screenshot for the above all functionalities are attached below: -

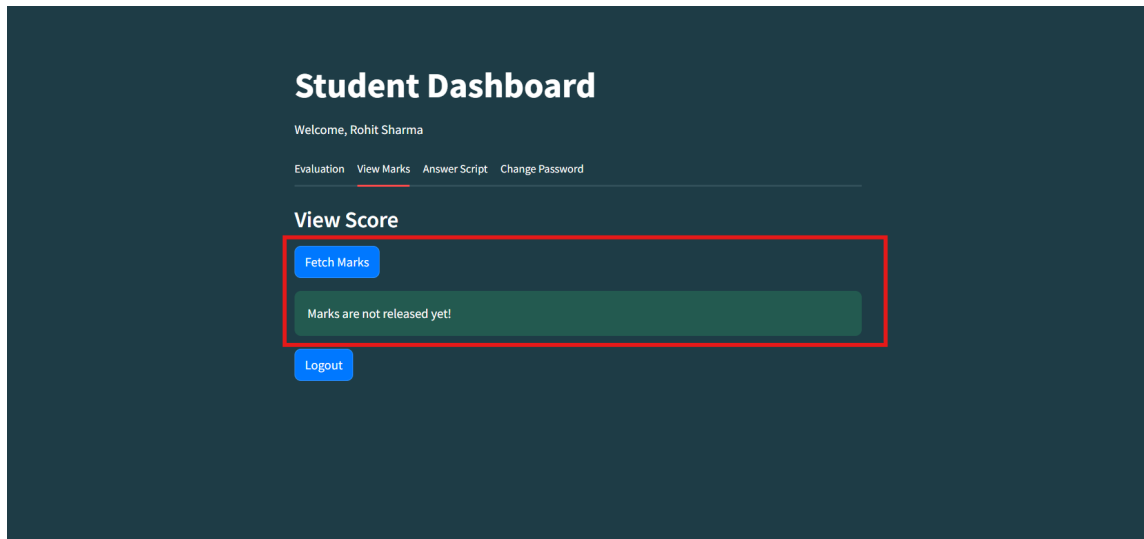


In the above screenshot we can clearly see that the “Evaluation Files” link is available and the student can use the link to open the files that he needs to evaluate.

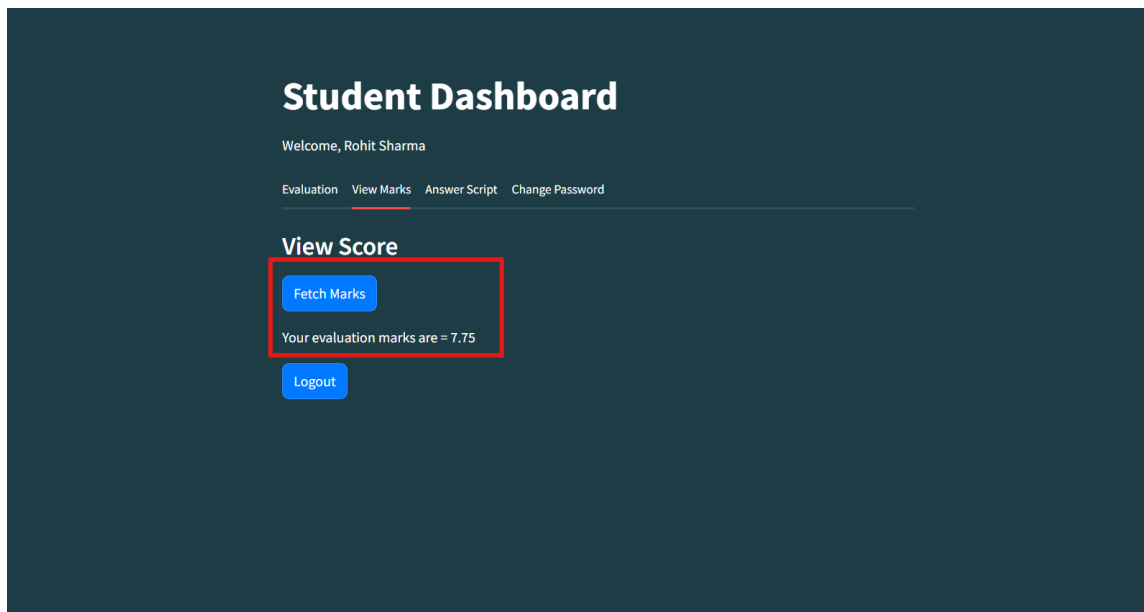


In the above screenshot we can see that the “Evaluation Sheet” link is available and the student can use the link to open the sheet in which he has to update the marks of their peers.

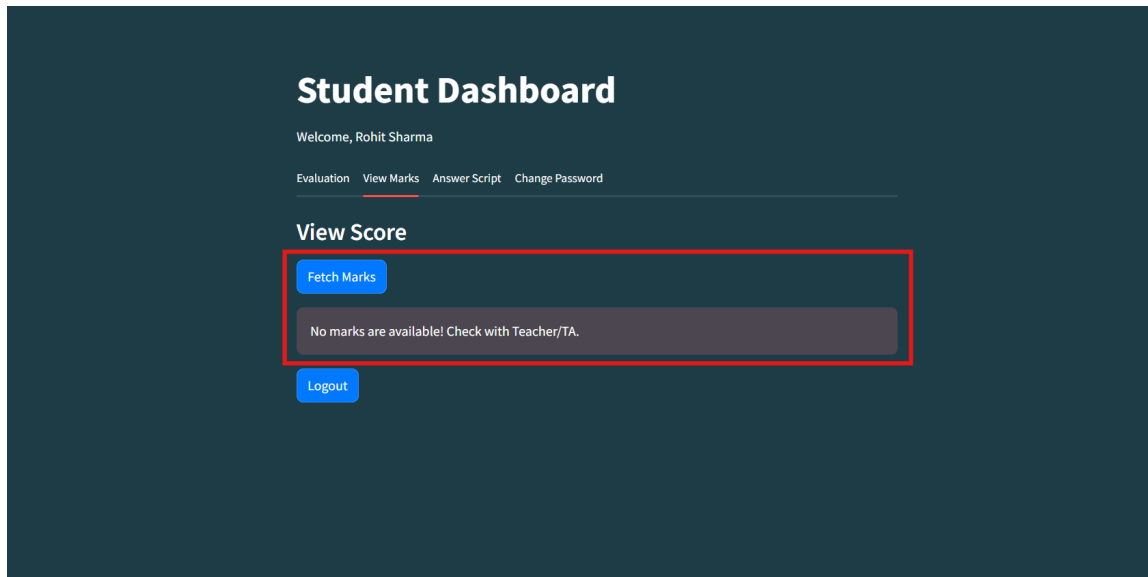
- **Student dashboard ( View Marks) window: -**



The “Fetch Marks” button on the View Marks window will help the student to view their marks for the current Quiz/Test. If the marks are not released yet then they will get the above message on their portal.

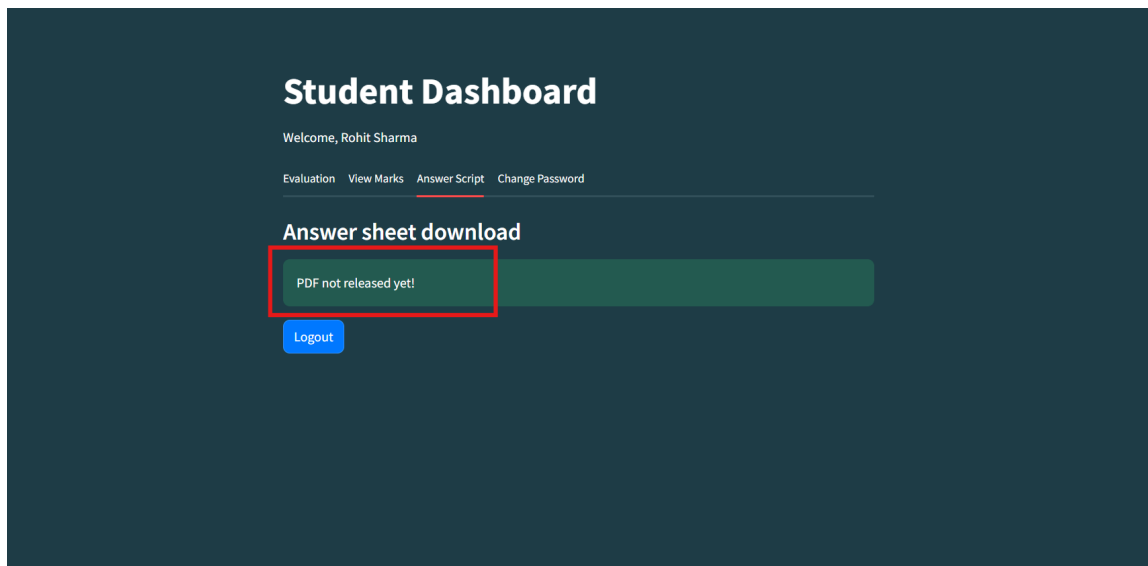


If the marks are released and available for the students, then they can see their marks with a display message as shown above.

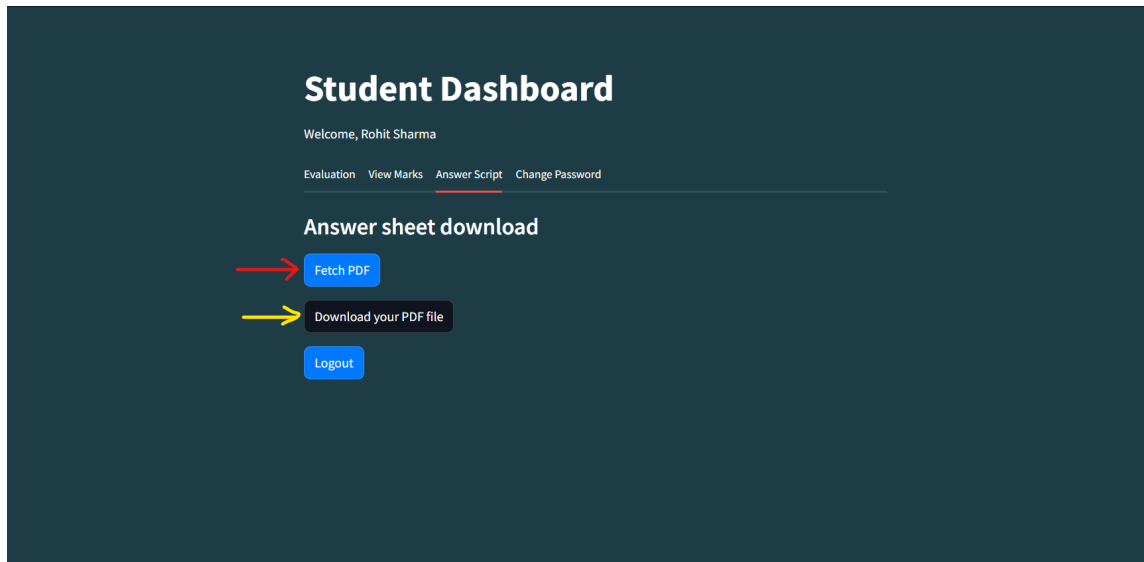


If the marks are released but the marks are not available for the students, then they will see the above message which asks them to get in touch with the teacher or TA of the subject.

- **Student dashboard ( Answer Script) window: -**



If the Answer sheet is not released that means the evaluation is still pending then the students will see the above message which says “PDF not released yet!”.



If the Answer sheet is available, that is the evaluation is done, then they can use the “Fetch PDF” button depicted with red arrow to get the “Download your PDF File” option depicted with yellow arrow. Using the above “Download your PDF File” option the student can download their Answer scripts and can view them.

The “Change Password” functionality is the same for all the users and the students can read the instructions for password change from the above “Change Password window” section of the instructions.

## REPLICATION

If we want to store the data from the previous test we need to save the copy of the target folder from the administrator side to the different google drive folder in order to clear the data and run the portal for new data.

## Annexure-II (Code & Implementation of Peer Evaluation)

The code is divided into two parts, the first part is the Python code which creates the front end and handles the all user inputs and the second part is the App Script code that takes care of the all backend process.

### Python files: -

#### 1. App.py -

```
import io
import re
import random
import smtplib
import time
import json
import bcrypt
import gspread
import requests
import streamlit as st
import plotly.express as px
import pandas as pd
from Rename_File import process_pdfs_in_folder # process_signatures,
load_stored_signatures
from googleapiclient.discovery import build
from googleapiclient.http import MediaIoBaseUpload
from googleapiclient.http import MediaIoBaseDownload
from googleapiclient.errors import HttpError
from oauth2client.service_account import ServiceAccountCredentials
from email.mime.text import MIMEText

# Google Sheets and Google Drive setup
SCOPE = [
    "https://spreadsheets.google.com/feeds",
    "https://www.googleapis.com/auth/drive"
```

```
]
```

```
CREDENTIALS_FILE = "peer-evaluation-440806-5b8bd496fe1e.json"
```

```
SHEET_NAME = "UserRoles"
```

```
web_app_url
```

```
=
```

```
"https://script.google.com/macros/s/AKfycbz15j0rCy0W0nqO6PAikwoycMyJTnUDDO1pc  
eE1sjiMbHTDebMdV0qgIhL6fp0Ni6Vx/exec"
```

```
# Initialize connection to Google Sheets
```

```
def connect_to_google_sheets():
```

```
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,  
SCOPE)
```

```
    client = gspread.authorize(creds)
```

```
    sheet = client.open(SHEET_NAME).sheet1
```

```
    return sheet
```

```
# Google Drive authentication
```

```
def authenticate_drive():
```

```
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,  
SCOPE)
```

```
    service = build('drive', 'v3', credentials=creds)
```

```
    return service
```

```
# Fetch users from Google Sheets
```

```
def get_users_from_sheets():
```

```
    sheet = connect_to_google_sheets()
```

```
    records = sheet.get_all_records()
```

```
    return records
```

```
# Function to generate a random 6-digit OTP
```

```

def generate_otp():
    return str(random.randint(100000, 999999))

# Function to send OTP to email
def send_otp_email(otp, recipient_email, name):
    sender_email = "aivs-coordinator@iitrpr.ac.in" # Replace with your email
    sender_password = "tahx kkic mygc zjni" # Replace with your email password or
    app-specific password

    # Create the email content
    message = MIMEText(f"Hello {name},\n\n Your Registration OTP is: {otp}.\n\n Thanks
    & Regards,\nCSE, IIT Ropar")
    message["Subject"] = "OTP for Registration"
    message["From"] = sender_email
    message["To"] = recipient_email

    try:
        # Connect to SMTP server and send email
        server = smtplib.SMTP_SSL("smtp.gmail.com", 465)
        server.login(sender_email, sender_password)
        response = server.sendmail(sender_email, recipient_email, message.as_string())
        server.quit()

        if response: # If response contains something, it means there was an issue
            print(f"Error: {response}")
            return False
        else:
            print("Email sent successfully!")
            return True # Email sent successfully
    except smtplib.SMTPRecipientsRefused:
        # Handle case where recipient email is invalid or not found
        print(f"Error: Address not found for recipient: {recipient_email}")
        return -1 # Return -1 if the recipient email is invalid
    except Exception as e:

```



```
print(f"Error sending OTP email: {e}")
return False
```

```
def validate_password(password):
    pattern = re.compile(r'^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$')
    return pattern.match(password)
```

```
# Add new user to Google Sheets with role auto-assignment
```

```
def register_user(username, password, name):
```

```
    sheet = connect_to_google_sheets()
```

```
    # Check if the email contains numeric values (assumed to be student)
```

```
    if re.search(r'\d', username) or not username.endswith("@iitrpr.ac.in"):
```

```
        role = "Student"
```

```
    else:
```

```
        role = "Teacher"
```

```
    # Hash the password before saving
```

```
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
```

```
    new_user = [username, hashed_password.decode('utf-8'), role, name]
```

```
    # new_user = [username, password, role, name]
```

```
    sheet.append_row(new_user)
```

```
    return role
```

```
def update_role_to_Teacher(username):
```

```
    sheet = connect_to_google_sheets()
```

```
    records = sheet.get_all_records()
```

```
    for i, user in enumerate(records, start=2): # start=2 to account for 1-based index in
Google Sheets
```

```

        if user['username'] == username: #and user['role'] == 'Student':
            sheet.update_cell(i, 3, 'Teacher') # Assuming role is in column 3
            return True
        return False

# Update role from Student to TA (only for Teachers)
def update_role_to_ta(username):
    sheet = connect_to_google_sheets()
    records = sheet.get_all_records()
    for i, user in enumerate(records, start=2): # start=2 to account for 1-based index in
Google Sheets
        if user['username'] == username and user['role'] == 'Student':
            sheet.update_cell(i, 3, 'TA') # Assuming role is in column 3
            return True
        return False

# Update role from TA to Student (only for Teachers)
def update_role_to_Student(username):
    sheet = connect_to_google_sheets()
    records = sheet.get_all_records()
    for i, user in enumerate(records, start=2): # start=2 to account for 1-based index in
Google Sheets
        if user['username'] == username: #and user['role'] == 'TA':
            sheet.update_cell(i, 3, 'Student') # Assuming role is in column 3
            return True
        return False

# Verify user credentials
def login(username, password, users):
    for user in users:
        if user['username'] == username:
            # Check if the password matches the stored hash

```

```

        if bcrypt.checkpw(password.encode('utf-8'), user['password'].encode('utf-8')):
            st.session_state["login_status"] = True
            st.session_state["role"] = user["role"]
            st.session_state["username"] = username
            st.session_state["page"] = "dashboard"
            st.session_state["message"] = None
            st.session_state["name"] = user["name"]
            return
        else:
            st.error("Incorrect Password!")
            time.sleep(2)
            st.rerun()
            return
    st.error("Incorrect Username or Password!")
    time.sleep(2)
    st.rerun()

```

# Logout function

```

def logout():
    st.session_state["login_status"] = False
    st.session_state["role"] = None
    st.session_state["username"] = None
    st.session_state["name"] = None
    st.session_state["page"] = "login"
    st.success("Logging out!")
    time.sleep(0.5)
    # st.session_state["message"] = "Logged out successfully"

```

# Function to change password

```

def change_password(username, current_password, new_password):
    sheet = connect_to_google_sheets()
    records = sheet.get_all_records()

```

```

# Find the user in the records
for i, user in enumerate(records, start=2): # start=2 for 1-based indexing (Google
Sheets)
    if user['username'] == username:
        # Check if the current password matches the stored hash
        if bcrypt.checkpw(current_password.encode('utf-8'),
user['password'].encode('utf-8')):
            # Hash the new password
            hashed_new_password = bcrypt.hashpw(new_password.encode('utf-8'),
bcrypt.gensalt()).decode('utf-8')

            # Update the password in the sheet
            sheet.update_cell(i, 2, hashed_new_password)
            return True # Password changed successfully
        else:
            return False # Current password is incorrect
    return False # User not found

```

```

def change_password_dashboard():
    st.header("Change Password")

    current_password = st.text_input("Current Password", type="password")
    new_password = st.text_input("New Password", type="password")
    confirm_password = st.text_input("Confirm New Password", type="password")

    if st.button("Change Password"):
        if new_password != confirm_password:
            st.error("New password and confirm password do not match!")
        elif not validate_password(new_password):
            st.error(

```

```
        "Password must include at least: - \n1. One uppercase letter. \n2. One lowercase letter. \n3. One special character. \n4. One numerical digit. \n5. Must be at least 8 characters long.")
```

```
    else:
        success = change_password(st.session_state['username'], current_password,
new_password)
        if success:
            st.success("Password changed successfully!")
            time.sleep(2)
            logout()
            st.rerun()
        else:
            st.error("Failed to change password. Incorrect current password.")
```

```
def trigger_google_apps_script(function_name):
```

```
    url = f"{web_app_url}?action={function_name}" # Append the function name as the 'action' parameter
```

```
    try:
        response = requests.get(url)
        if response.status_code == 200:
            st.success(f"{function_name} executed successfully!")
        else:
            st.error(f"Failed to execute {function_name}. Status code: {response.status_code}")
    except Exception as e:
        st.error(f"An error occurred: {str(e)}")
```

```
def send_data_to_script(param_value):
```

```
    url = f"{web_app_url}?action={'SetParameter'}"
    try:
        # Use GET for simple parameters
```

```

response = requests.get(url, params={'param': param_value})

if response.status_code == 200:
    st.success(f"Response from Google Apps Script: {response.text}")
else:
    st.error(f"Failed to communicate with Apps Script: {response.status_code}")
except Exception as e:
    st.error(f"Error: {e}")

```

# Function to check if a file already exists in Google Drive folder

```

def file_exists(drive_service, folder_id, file_name):
    query = f"'{folder_id}' in parents and name='{file_name}'"
    results = drive_service.files().list(q=query, spaces='drive', fields='files(id,
name)').execute()
    files = results.get('files', [])
    return any(file['name'] == file_name for file in files)

```

# Function to upload PDF files to Google Drive

```

def upload_pdfs(uploaded_files, folder_id):
    drive_service = authenticate_drive()
    count = 0

    for uploaded_file in uploaded_files:
        if file_exists(drive_service, folder_id, uploaded_file.name):
            # st.warning(f"PDF file '{uploaded_file.name}' already exists in the folder.")
            continue

        file_metadata = {
            'name': uploaded_file.name,
            'parents': [folder_id]

```

```

    }
    media = MedialoBaseUpload(uploaded_file, mimetype='application/pdf')
    drive_service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
    count = count + 1
    # st.session_state["success_message"] = f"Uploaded PDF file
'{uploaded_file.name}' to Google Drive"

```

```

st.success(f" The {count} files are uploaded to the Google Drive.")

```

```

# Function to upload Google Sheets files to Google Drive

```

```

def upload_sheets(uploaded_file, folder_id):

```

```

    drive_service = authenticate_drive()

```

```

    if file_exists(drive_service, folder_id, uploaded_file.name):

```

```

        st.warning(f"Google Sheet file '{uploaded_file.name}' already exists in the folder.")

```

```

        return

```

```

    file_metadata = {

```

```

        'name': uploaded_file.name,

```

```

        'parents': [folder_id]

```

```

    }

```

```

# Upload the file to Google Drive

```

```

    media = MedialoBaseUpload(uploaded_file, mimetype=uploaded_file.type,
resumable=True)

```

```

    uploaded = drive_service.files().create(body=file_metadata, media_body=media,
fields='id').execute()

```

```

st.success("The Excel sheet has been uploaded to the Google Drive.")

```

```

def delete_all_from_folder(folder_id):

```

```

    """

```

Deletes all files and subfolders from a Google Drive folder.

"""

```
service = authenticate_drive() # Authenticate using the provided function
```

```
try:
```

```
    # List all files and folders in the given folder
```

```
    query = f'"{folder_id}" in parents'
```

```
    results = service.files().list(q=query, fields="files(id, name, mimeType)").execute()
```

```
    items = results.get('files', [])
```

```
    if not items:
```

```
        st.error("The folder is already empty!")
```

```
        return
```

```
    for item in items:
```

```
        file_id = item['id']
```

```
        file_name = item['name']
```

```
        mime_type = item['mimeType']
```

```
    try:
```

```
        # Delete the file or folder
```

```
        service.files().delete(fileId=file_id).execute()
```

```
    except HttpError as error:
```

```
        st.error(f"An error occurred while deleting {file_name}: {error}")
```

```
    st.success(f"All data has been deleted.")
```

```
except HttpError as error:
```

```
    st.error(f"An error occurred: {error}")
```

```
# Helper function to connect to a specific Google Sheet
```

```
def connect_to_google_sheets_with_name(sheet_name):
```



```

        creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
        client = gspread.authorize(creds)
        sheet = client.open(sheet_name)
        return sheet

```

```

def connect_to_google_sheets_with_id(file_id):
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
SCOPE)
    client = gspread.authorize(creds)
    sheet = client.open_by_key(file_id)
    return sheet

```

```

def get_student_details(username):
    # Connect to the specific Google Sheet containing details
    sheet_name = "UI/UX Peer Evaluation"
    sheet = connect_to_google_sheets_with_name(sheet_name) # Modify to accept a
sheet name
    peer_eval_sheet = sheet.worksheet('PeerEval') # Open the "PeerEval" sheet

    # Fetch all the data from the "PeerEval" sheet
    records = peer_eval_sheet.get_all_records()

    # Check if data exists
    if not records or 'Assigned Folder Link' not in records[0] or 'Spreadsheet Link' not in
records[0]: # Checking the first row (header) for the column
        return -1, -1, -1 # Return None if the column does not exist

    # Find marks for the current user
    for record in records:
        if record['EMail ID'] == username: # Ensure this matches your column name

```

```
        return record['Unique ID'], record['Assigned Folder Link'], record['Spreadsheet Link'] # Returning the Unique id
```

```
    return None, None, None # If no details found for the user
```

```
def get_student_marks(username):
```

```
    # Connect to the specific Google Sheet containing marks
```

```
    sheet_name = "UI/UX Peer Evaluation"
```

```
    sheet = connect_to_google_sheets_with_name(sheet_name) # Modify to accept a sheet name
```

```
    peer_eval_sheet = sheet.worksheet('PeerEval') # Open the "PeerEval" sheet
```

```
    # Fetch all the data from the "PeerEval" sheet
```

```
    records = peer_eval_sheet.get_all_records()
```

```
    # Check if 'Average Marks' column exists
```

```
    if not records or 'Average Marks' not in records[0]: # Checking the first row (header) for the column
```

```
        # print("Average Marks column not found")
```

```
        return -1 # Return None if the column does not exist
```

```
    # Find marks for the current user
```

```
    for record in records:
```

```
        if record['EMail ID'] == username and record['Average Marks']: # Ensure this matches your column name
```

```
            return record['Average Marks'] # Returning the Average Mark's
```

```
    return None # If no marks found for the user
```

```
# Fetch the student's PDF from Google Drive using unique ID
```

```
def get_student_pdf(unique_id):
```

```
    drive_service = authenticate_drive()
```

```

folder_id = "1tsH69oimECrQwaFq58i9Hyh_iHjCJyjE"
query = f"'{folder_id}' in parents and name contains '{unique_id}'"
results = drive_service.files().list(q=query, fields="files(id, name)").execute()
files = results.get('files', [])

```

```

if files:

```

```

    file_id = files[0]['id']
    file_name = files[0]['name']

```

```

    # Download the PDF

```

```

    request = drive_service.files().get_media(fileId=file_id)

```

```

    fh = io.BytesIO()

```

```

    downloader = MediaIoBaseDownload(fh, request)

```

```

    done = False

```

```

    while not done:

```

```

        status, done = downloader.next_chunk()

```

```

    fh.seek(0)

```

```

    return fh, file_name

```

```

return None, None

```

```

def renaming_files():

```

```

    folder_id = '1tsH69oimECrQwaFq58i9Hyh_iHjCJyjE' # Replace with your Google
    Drive folder ID

```

```

    num_files = process_pdfs_in_folder(folder_id)

```

```

    return num_files

```

```

    # Authenticate Google Drive

```

```

    # service = authenticate_drive()

```

```

    # Google Drive folder IDs

```

```

    # stored_signatures_folder_id = '14QLNPdIRUZ3ici-GePoEewUCmxemjhUD' #The
    folder where we want to keep the Stored signature

```

```
# uploaded_signatures_folder_id = '1ORVrU-UoXyDS-1ovyuk7FAjb_p94gnsx' #This
will be the folder where our pdf files are kept
```

```
# destination_folder_id = '1bPhLMZONpsPDxM9z_vQD2J9jAjtWg3FG' # Folder
where renamed files will be moved
```

```
# Load stored signatures directly from Google Drive
# stored_signatures, stored_filenames = load_stored_signatures(service,
stored_signatures_folder_id)
```

```
# Process uploaded signatures and copy renamed files to the destination folder
# matched_files = process_signatures(service, uploaded_signatures_folder_id,
stored_signatures, stored_filenames, destination_folder_id)
```

```
# print(f"Matching process completed. Total matched files: {len(matched_files)}")
```

```
# Connect to Google Sheets
```

```
def fetch_sheet_data(sheet_name, worksheet_name):
```

```
    # gc = gspread.service_account(filename=CREDENTIALS_FILE)
```

```
    # spreadsheet = gc.open(sheet_name)
```

```
    # worksheet = spreadsheet.worksheet(worksheet_name)
```

```
    # return worksheet.get_all_values()
```

```
    creds = ServiceAccountCredentials.from_json_keyfile_name(CREDENTIALS_FILE,
                                                              SCOPE) # Replace with your credentials file
```

```
    client = gspread.authorize(creds)
```

```
    sheet = client.open(sheet_name).worksheet(worksheet_name) # Open the specific
worksheet
```

```
    return sheet
```

```
# Fetch Peer Average and Standard Deviation data
```

```
def get_peer_average_data(sheet):
```

```
    data = sheet.get_all_values()
```

```

for index, row in enumerate(data):
    if row[0] == "Peer": # Check the first column of each row
        st_idx = index
    if row[0] == "SD": # Check the first column of each row
        std_idx = index
    if row[0] == "Peer Average": # Check the first column of each row
        pavg_idx = index

# Extract student IDs, Standard Deviations, and Peer Averages
student_ids = data[st_idx][1:] # First row has student IDs starting from B1
std_devs = list(map(float, data[std_idx][1:])) # Row 6 has Standard Deviations starting
from B6
peer_averages = list(map(float, data[pavg_idx][1:])) # Row 7 has Peer Averages
starting from B7

# Create a DataFrame with relevant data
df = pd.DataFrame({
    'Student': student_ids,
    'Standard Deviation': std_devs,
    'Average Score': peer_averages
})

return df

# Generate Standard Deviation Chart
def create_standard_deviation_chart(df):
    fig = px.bar(df, x='Student', y='Standard Deviation', title='Standard Deviation of Scores
for Each Student',
        color_discrete_sequence=['#4682B4'])
    fig.update_layout(yaxis_title='Standard Deviation', xaxis_title='Student',
        plot_bgcolor='white', paper_bgcolor='white', # Set background colors to
white
        font_color='black', title_font_color='black',

```

```

        xaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black')),
        yaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black'))
    )
    return fig

# Generate Peer Average Scores Chart
def create_average_scores_chart(df):
    fig = px.bar(df, x='Student', y='Average Score', title='Peer Average Scores for Each Student',
                 color_discrete_sequence=['#FF6347'])
    fig.update_layout(yaxis_title='Peer Average', xaxis_title='Student',
                      plot_bgcolor='white', paper_bgcolor='white', # Set background colors to white
                      font_color='black', title_font_color='black',
                      xaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black')),
                      yaxis=dict(title_font=dict(color='black'), tickfont=dict(color='black'))
    )
    return fig

# Trigger when "Generate Charts" button is clicked
def generate_charts():
    sheet = fetch_sheet_data('UI/UX Peer Evaluation', 'Evaluation Results')
    df = get_peer_average_data(sheet) # Get the data
    # Display two charts
    st.plotly_chart(create_standard_deviation_chart(df))
    st.plotly_chart(create_average_scores_chart(df))

# Function to convert Excel file to Google Sheets
def convert_excel_to_google_sheet(file_id):
    drive_service = authenticate_drive()

```

```

# File metadata for conversion
file_metadata = {
    'mimeType': 'application/vnd.google-apps.spreadsheet'
}

# Convert the file and get the new file ID
converted_file = drive_service.files().copy(fileId=file_id, body=file_metadata).execute()
return converted_file['id']

# Function to list all files in the given folder (Google Drive)
def list_spreadsheets_in_folder(folder_id):
    service = authenticate_drive() # Authenticate Drive using the new function
    query = f'"{folder_id}" in parents and
(mimeType='application/vnd.google-apps.spreadsheet' OR
mimeType='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')"
    results = service.files().list(q=query).execute()
    files = results.get('files', [])

    if not files:
        print('No spreadsheets found in the folder.')
        return []

    return files

# Function to delete only spreadsheet files in the specified folder
def delete_all_spreadsheets_in_folder(folder_id):
    drive_service = authenticate_drive()

    # List all files in the folder
    query = f'"{folder_id}" in parents and trashed = false'
    results = drive_service.files().list(q=query).execute()
    files = results.get('files', [])

```

```

# Iterate through files and delete only those with the MIME type for Google Sheets
for file in files:
    if file['mimeType'] == 'application/vnd.google-apps.spreadsheet' or file['mimeType']
    == 'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet':
        try:
            drive_service.files().delete(fileId=file['id']).execute()
            print(f"Deleted spreadsheet: {file['name']}")
        except Exception as e:
            print(f"Error deleting spreadsheet {file['name']}: {e}")
    else:
        print(f"Skipped non-spreadsheet file: {file['name']}")

```

```

# Function to copy data from each spreadsheet to the target sheet
def copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name):
    # Get the target spreadsheet using the new function
    target_spreadsheet =
connect_to_google_sheets_with_name(target_spreadsheet_name)

    try:
        target_worksheet = target_spreadsheet.worksheet(target_worksheet_name)
    except gspread.WorksheetNotFound:
        #print(f"Worksheet '{target_worksheet_name}' not found in the target spreadsheet.")
        return

```

```

target_worksheet.clear()
#print("Target worksheet cleared successfully.")

```

```

# List all spreadsheets in the folder
files = list_spreadsheets_in_folder(folder_id)

```

```

for file in files:

```



```

file_id = file['id']
file_name = file['name']
mime_type = file['mimeType']

#print(f"Processing file: {file_name}")

# If it's an Excel file, convert it to a Google Sheet
if mime_type ==
'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet':
    file_id = convert_excel_to_google_sheet(file_id)

# Now open the file as a Google Sheet

source_spreadsheet = connect_to_google_sheets_with_id(file_id)
source_worksheet = source_spreadsheet.get_worksheet(0) # Assuming data is in
the first worksheet
source_data = source_worksheet.get_all_values()

# Append data to the target worksheet
if source_data:
    target_worksheet.append_rows(source_data, value_input_option="RAW")
    st.success("Data copied to PeerEval successfully.")
    return
else:
    st.error("Error copying data to PeerEval!")

#print("Data copied to PeerEval worksheet successfully.")
#delete_all_spreadsheets_in_folder(folder_id)
#print("Spreadsheet is removed successfully")

# Function to clear all worksheet data except "Cumulative Marks"
def reset_worksheets(spreadsheet_name, keep_sheet_name):
    # Open the target spreadsheet

```

```
spreadsheet = connect_to_google_sheets_with_name(spreadsheet_name)
```

```
for worksheet in spreadsheet.worksheets():
```

```
    # Check if worksheet name is not "Cumulative Marks"
```

```
    if worksheet.title != keep_sheet_name:
```

```
        # Clear the content of the worksheet
```

```
        worksheet.clear()
```

```
print(f"All worksheets except {keep_sheet_name} have been cleared.")
```

```
# Function to delete all worksheets except the specified one
```

```
def delete_other_worksheets(target_spreadsheet_name, keep_sheet_name):
```

```
    spreadsheet = connect_to_google_sheets_with_name(target_spreadsheet_name)
```

```
    try:
```

```
        # List all worksheets in the spreadsheet
```

```
        all_worksheets = spreadsheet.worksheets()
```

```
        # Loop through each worksheet
```

```
        for sheet in all_worksheets:
```

```
            # Check if the worksheet is not the one to keep
```

```
            if sheet.title != keep_sheet_name:
```

```
                # Delete the worksheet
```

```
                spreadsheet.del_worksheet(sheet)
```

```
                print(f"Deleted worksheet: {sheet.title}")
```

```
    print(f"All worksheets except '{keep_sheet_name}' have been deleted.")
```

```
except Exception as e:
```

```
    print(f"Error while deleting worksheets: {e}")
```

```
def admin_dashboard():
```

```
    st.title("Admin Dashboard")
```

```
    st.write(f"Welcome, {st.session_state['name']}")
```

```
# Create tabs for each action
tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8, tab9 = st.tabs(
    ["Upload Data", "Rename Files", "Role Manager", "Pre Evaluation", "Reminder
    Mail", "Post Evaluation",
    "Generate Charts", "Send Marks", "Cumulative Score", "Clear Files", "Change
    Password"])
```

```
# Tab for File upload option
with tab:
    # Folder ID for the Google Drive folder where the files will be saved
    folder_id = "1tsH69oimECrQwaFq58i9Hyh_iHjCJyJE" # Replace this with your
    folder ID
```

```
# Allow file upload for multiple Google Sheets
st.subheader("Upload Student Information")
    sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"],
    accept_multiple_files=False,
    key="sheet_uploader")
```

```
if sheet_files:
    upload_sheets(sheet_files, folder_id)
```

```
# Allow file upload for multiple PDFs
st.subheader("Upload PDF Files")
    pdf_files = st.file_uploader("Upload PDF files", type=["pdf"],
    accept_multiple_files=True, key="pdf_uploader")
```

```
if pdf_files:
    upload_pdfs(pdf_files, folder_id)
    time.sleep(2)
    logout()
    st.rerun()
```

```

st.subheader("Clear Files for Re-Uploading")
if st.button("Clear Data"):
    delete_all_from_folder(folder_id)

with tab0:
    st.subheader("Rename PDF Files")
    if st.button("Rename Files"):
        count = renaming_files()
        if count == 0:
            st.error(f"No file for renaming has been found.")
        else:
            st.success(f"A total of {count} files are renamed.")

# Tab for TA update
with tab1:
    student_username = st.text_input("Enter Username")
    st.subheader("Assign Teacher")
    if st.button("Update Role to Teacher"):
        if update_role_to_Teacher(student_username):
            st.success(f"The role updated to Teacher.")
        else:
            st.error("Failed to update the role. Check if the username exists.")
    st.subheader("Assign TA")
    if st.button("Update Role to TA"):
        if update_role_to_ta(student_username):
            st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to TA.")
        else:
            st.error("Failed to update the role. Check if the username exists and belongs
to a student.")
    st.subheader("Assign Student")
    if st.button("Update Role to Student"):
        if update_role_to_Student(student_username):
            st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
Student.")

```

```

else:
    st.error("Failed to update the role. Check if the username exists.")

# Tab for Pre Evaluation
with tab2:
    st.subheader("Set Parameters")
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        send_data_to_script(num_Questions)

    target_spreadsheet_name = "UI/UX Peer Evaluation"
    # target_spreadsheet_name = "Sample_Run"
    target_worksheet_name = "PeerEval"
    st.subheader("Extract Student Information")
    if st.button("Data Extraction"):
        copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        delete_all_spreadsheets_in_folder(folder_id)
    st.subheader("Run Pre evaluation")
    if st.button("Pre Evaluation"):
        trigger_google_apps_script("PreEval")

# Tab for Checking Pending Evaluations
with tab3:
    st.subheader("Send Reminder Mail")
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")

# Tab for Post Evaluation
with tab4:
    st.subheader("Run Post evaluation")
    if st.button("Post Evaluation"):

```

```
trigger_google_apps_script("PostEval")
```

```
# Tab for Generating Charts
```

```
with tab5:
```

```
    st.subheader("View Average and SD chart")
```

```
    if st.button("Generate Charts"):
```

```
        generate_charts()
```

```
        # trigger_google_apps_script("GenChart")
```

```
# Tab for Sending Marks
```

```
with tab6:
```

```
    st.subheader("Share Marks with Students")
```

```
    if st.button("Send Marks"):
```

```
        trigger_google_apps_script("SendMail")
```

```
with tab7:
```

```
    st.subheader("Total Score")
```

```
    if st.button("Send Cumulative Score"):
```

```
        trigger_google_apps_script("SendFinalM")
```

```
with tab8:
```

```
    st.subheader("Reset files for new Quiz/Exam/Test")
```

```
    if st.button("Reset Files"):
```

```
        reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
```

```
        delete_all_from_folder(folder_id)
```

```
    st.subheader("Delete all data")
```

```
    if st.button("Clear Files"):
```

```
        delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)
```

```
        delete_all_from_folder(folder_id)
```

```
with tab9:
```

```
    change_password_dashboard()
```

```

def teacher_dashboard():
    st.title("Teacher Dashboard")
    # var_user = st.session_state['username'].split('@')[0]
    # if '.' in var_user:
    #     st.write(f"Welcome, Dr. {var_user.split('.')[0].capitalize()}")
    # else:
    #     st.write(f"Welcome, Dr. {var_user.capitalize()}")
    st.write(f"Welcome, {st.session_state['name']}")

    # Create tabs for each action
    tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6, tab7, tab8 = st.tabs(
        ["Rename Files", "Role Manager", "Pre Evaluation", "Reminder Mail", "Post
Evaluation",
        "Generate Charts", "Send Marks", "Cumulative Score", "Clear Files", "Change
Password"])

    # Tab for TA update
    with tab:
        st.subheader("Rename PDF Files")
        if st.button("Rename Files"):
            count = renaming_files()
            if count == 0:
                st.error(f"No file for renaming has been found.")
            else:
                st.success(f"A total of {count} files are renamed.")

    with tab0:
        student_username = st.text_input("Enter Student's Username")
        st.subheader("Assign TA")
        if st.button("Update Role to TA"):
            if update_role_to_ta(student_username):
                st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to TA.")
            else:

```

```

        st.error("Failed to update the role. Check if the username exists and belongs
to a student.")
    st.subheader("Assign Student")
    if st.button("Update Role to Student"):
        if update_role_to_Student(student_username):
            st.success(f"{student_username.split('.')[0].capitalize()}'s role updated to
Student.")
        else:
            st.error("Failed to update the role. Check if the username exists.")

# Tab for Pre Evaluation
with tab1:
    st.subheader("Set Parameters")
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        send_data_to_script(num_Questions)

    folder_id = "1tsH69oimECrQwaFq58i9Hyh_iHjCJyjE"
    target_spreadsheet_name = "UI/UX Peer Evaluation"
    # target_spreadsheet_name = "Sample_Run"
    target_worksheet_name = "PeerEval"
    st.subheader("Extract Student Information")
    if st.button("Data Extraction"):
        copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        delete_all_spreadsheets_in_folder(folder_id)
    st.subheader("Run Pre evaluation")
    if st.button("Pre Evaluation"):
        # copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        trigger_google_apps_script("PreEval")

```



# Tab for Checking Pending Evaluations

with tab2:

```
st.subheader("Send Reminder Mail")
if st.button("Reminder Mail"):
    trigger_google_apps_script("CheckEval")
```

# Tab for Post Evaluation

with tab3:

```
st.subheader("Run Post evaluation")
if st.button("Post Evaluation"):
    trigger_google_apps_script("PostEval")
```

# Tab for Generating Charts

with tab4:

```
st.subheader("View Average and SD chart")
if st.button("Generate Charts"):
    generate_charts()
    # trigger_google_apps_script("GenChart")
```

with tab5:

```
st.subheader("Share Marks with Students")
if st.button("Send Marks"):
    trigger_google_apps_script("SendMail")
```

with tab6:

```
st.subheader("Total Score")
if st.button("Send Cumulative Score"):
    trigger_google_apps_script("SendFinalM")
```

with tab7:

```
st.subheader("Reset files for new Quiz/Exam/Test")
if st.button("Reset Files"):
    reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
    delete_all_from_folder(folder_id)
```

```

    st.subheader("Delete all data")
    if st.button("Clear Files"):
        delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)
        delete_all_from_folder(folder_id)

with tab8:
    change_password_dashboard()

# Role-based content: Teacher Dashboard with multiple file uploads
def ta_dashboard():
    st.title("TA Dashboard")
    # st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")
    st.write(f"Welcome, {st.session_state['name']}")

# Create tabs for each action
tab, tab0, tab1, tab2, tab3, tab4, tab5, tab6 = st.tabs(
    ["Upload Data", "Rename Files", "Pre Evaluation", "Reminder Mail", "Post
Evaluation", "Generate Charts",
    "Clear Files", "Change Password"])

# Tab for File upload option
with tab:
    # Folder ID for the Google Drive folder where the files will be saved
    folder_id = "1tsH69oimECrQwaFq58i9Hyh_iHjCJyjE" # Replace this with your
folder ID

# Allow file upload for multiple Google Sheets
st.subheader("Upload Student Information")
    sheet_files = st.file_uploader("Upload Google Sheet", type=["xlsx"],
accept_multiple_files=False,
    key="sheet_uploader")

if sheet_files:

```

```

upload_sheets(sheet_files, folder_id)

# Allow file upload for multiple PDFs
st.subheader("Upload PDF Files")
pdf_files = st.file_uploader("Upload PDF files", type=["pdf"],
accept_multiple_files=True, key="pdf_uploader")

if pdf_files:
    upload_pdfs(pdf_files, folder_id)
    time.sleep(2)
    logout()
    st.rerun()

st.subheader("Clear Files for Re-Uploading")
if st.button("Clear Data"):
    delete_all_from_folder(folder_id)

with tab0:
    st.subheader("Rename PDF Files")
    if st.button("Rename Files"):
        count = renaming_files()
        if count == 0:
            st.error(f"No file for renaming has been found.")
        else:
            st.success(f"A total of {count} files are renamed.")

# Tab for Pre Evaluation
with tab1:
    st.subheader("Set Parameters")
    num_Questions = st.number_input("Enter the number of questions", min_value=1,
max_value=100)
    # Button to submit the form
    if st.button("Set Parameters"):
        send_data_to_script(num_Questions)

```

```

folder_id = "1tsH69oimECrQwaFq58i9Hyh_iHjCJyjE"
target_spreadsheet_name = "UI/UX Peer Evaluation"
# target_spreadsheet_name = "Sample_Run"
target_worksheet_name = "PeerEval"
st.subheader("Extract Student Information")
if st.button("Data Extraction"):
    copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
    delete_all_spreadsheets_in_folder(folder_id)
    st.subheader("Run Pre evaluation")
    if st.button("Pre Evaluation"):
        # copy_data_to_peer_eval(folder_id, target_spreadsheet_name,
target_worksheet_name)
        trigger_google_apps_script("PreEval")

# Tab for Checking Pending Evaluations
with tab2:
    st.subheader("Send Reminder Mail")
    if st.button("Reminder Mail"):
        trigger_google_apps_script("CheckEval")

# Tab for Post Evaluation
with tab3:
    st.subheader("Run Post evaluation")
    if st.button("Post Evaluation"):
        trigger_google_apps_script("PostEval")

# Tab for Generating Charts
with tab4:
    st.subheader("View Average and SD chart")
    if st.button("Generate Charts"):
        generate_charts()
        # trigger_google_apps_script("GenChart")

```

```

with tab5:
    st.subheader("Reset files for new Quiz/Exam/Test")
    if st.button("Reset Files"):
        reset_worksheets(target_spreadsheet_name, "Cumulative Marks")
        delete_all_from_folder(folder_id)

    st.subheader("Delete all data")
    if st.button("Clear Files"):
        delete_other_worksheets(target_spreadsheet_name, target_worksheet_name)
        delete_all_from_folder(folder_id)

with tab6:
    change_password_dashboard()

def student_dashboard():
    st.title("Student Dashboard")
    # st.write(f"Welcome, {st.session_state['username'].split('.')[0].capitalize()}")
    st.write(f"Welcome, {st.session_state['name']}")

    # Creating tabs
    tab1, tab2, tab3, tab4 = st.tabs(["Evaluation", "View Marks", "Answer Script", "Change
Password"])

    # Tab for opening the peer evaluation spreadsheet
    with tab1:
        if st.session_state["username"]:
            # Fetch unique ID, and spreadsheet link using the session's username
            unique_id, folder_link, sheet_link =
get_student_details(st.session_state["username"])
        else:
            st.error("Username is Incorrect!")

```

```

t1, t2 = st.tabs(["Evaluation Files", "Evaluation Sheet"])
with t1:
    st.write("Link to Open Evaluation Files")
    if folder_link == -1:
        st.success("Nothing to Evaluate.")
    elif folder_link:
        st.markdown(f"[Evaluation Files]({folder_link})", unsafe_allow_html=True)
    else:
        st.error("Folder link not found.")
with t2:
    st.write("Link to Open Evaluation Sheets")
    if sheet_link == -1:
        st.success("Nothing to Evaluate.")
    elif sheet_link:
        st.markdown(f"[Evaluation Sheet]({sheet_link})", unsafe_allow_html=True)
    else:
        st.error("Spreadsheet link not found.")

```

# Tab for viewing marks

```

with tab2:
    st.subheader("View Score")
    if st.button("Fetch Marks"):
        # Fetch Marks
        marks = get_student_marks(st.session_state["username"])
        if marks and unique_id and marks != -1:
            st.write(f"Your evaluation marks are = {marks}")
        elif marks == -1:
            st.success("Marks are not released yet!")
        else:
            st.error("No marks are available!\nCheck with Teacher/TA.")

```

# Tab for downloading PDF

```

with tab3:
    st.subheader("Answer sheet download")

```

```

if unique_id == -1:
    st.success("PDF not released yet!")
else:
    if st.button("Fetch PDF"):
        pdf_file, file_name = get_student_pdf(unique_id)
        if pdf_file:
            st.download_button(
                label="Download your PDF file",
                data=pdf_file,
                file_name=file_name,
                mime='application/pdf'
            )
        else:
            st.error("PDF not found! Contact Teacher/TA.")

with tab4:
    change_password_dashboard()

```

# Main Streamlit app

```

def main():
    # Initialize session state variables if not present
    if "login_status" not in st.session_state:
        st.session_state["login_status"] = False
    if "role" not in st.session_state:
        st.session_state["role"] = None
    if "username" not in st.session_state:
        st.session_state["username"] = None
    if "page" not in st.session_state:
        st.session_state["page"] = "login"
    if "message" not in st.session_state:
        st.session_state["message"] = None
    if "success_message" not in st.session_state:
        st.session_state["success_message"] = None

```

```

if "name" not in st.session_state:
    st.session_state["name"] = None

# Set background color and input field styling using HTML
st.markdown(
    """
    <style>
    .stApp {
        background-color: #1f3f49; /* Light blue background */
    }
    .stTextInput>div>input, .stPasswordInput>div>input {
        background-color: white; /* White background for text and password inputs */
        color: black; /* Text color for input fields */
    }
    .stButton>button {
        background-color: #007bff; /* Optional: Style buttons with a color */
        color: white;
    }
    h1, h2, h3, h4, h5, h6, p {
        color: white;
    }
    </style>
    """,
    unsafe_allow_html=True
)

```

```

# Page routing based on session state

```

```

if st.session_state["page"] == "login":
    st.title("Peer Evaluation System")

```

```

# Tabs for Login and Registration

```

```

tab1, tab2 = st.tabs(["Login", "Register"])

```

```

with tab1:

```



```

st.header("Login")

with st.form(key='login_form'):
    username = st.text_input("Email ID")
    password = st.text_input("Password", type="password")
    submit_button = st.form_submit_button("Login")

    if submit_button:
        users = get_users_from_sheets()
        login(username, password, users)
        if st.session_state["login_status"]:
            st.rerun()

with tab2:
    st.header("Register")

    if "otp_sent" not in st.session_state:
        st.session_state["otp_sent"] = False # Flag to track if OTP is sent
    if "otp_verified" not in st.session_state:
        st.session_state["otp_verified"] = False # Flag to track if OTP is verified

    with st.form(key='register_form'):
        reg_name = st.text_input("Name", key='reg_name')
        reg_username = st.text_input("Email ID", key='reg_username')
        reg_password = st.text_input("Password", type="password",
key='reg_password')
        register_button = st.form_submit_button("Register")

        if register_button:
            # if not reg_username.endswith("@iitrpr.ac.in"):
            #     st.error("Email ID must be of @iitrpr.ac.in domain.")
            if not validate_password(reg_password):
                st.error(

```

"Password must include at least One: - \n1. Uppercase letter. \n2. Lowercase letter. \n3. Special character. \n4. Numerical digit. \n5. Must be at least 8 characters long.")

else:

var\_otp = generate\_otp()

if send\_otp\_email(var\_otp, reg\_username, reg\_name):

st.session\_state["otp\_sent"] = True

st.session\_state["otp"] = var\_otp

st.session\_state["username"] = reg\_username

st.session\_state["password"] = reg\_password

st.session\_state["name"] = reg\_name

st.success(f"OTP has been sent to the {reg\_username}")

elif send\_otp\_email(var\_otp, reg\_username, reg\_name) == -1:

st.error("Error: Email Address not found!")

else:

st.error("Error! OTP not sent.")

if st.session\_state.get("otp\_sent", False):

with st.form(key='verification\_form'):

reg\_otp = st.text\_input("OTP", key='reg\_otp')

verify\_button = st.form\_submit\_button("Verify")

if verify\_button:

if reg\_otp == st.session\_state["otp"]:

users = get\_users\_from\_sheets()

if any(user["username"] == st.session\_state["username"] for user in

users):

st.error("Username already exists")

else:

role = register\_user(st.session\_state["reg\_username"],

st.session\_state["password"], st.session\_state["name"])

st.success(f"User registered successfully with role: {role}")

time.sleep(2)

# Redirect to the login page

st.session\_state["page"] = "login"

st.rerun()

```

        else:
            st.error("Incorrect OTP!")

elif st.session_state["page"] == "dashboard":
    if st.session_state["role"] == "Admin":
        admin_dashboard()
    elif st.session_state["role"] == "Teacher":
        teacher_dashboard()
    elif st.session_state["role"] == "TA":
        ta_dashboard()
    elif st.session_state["role"] == "Student":
        student_dashboard()

# Logout button
if st.button("Logout"):
    logout()
    st.rerun()

if __name__ == "__main__":
    main()

```

## 2. Rename\_File.py -

```

from pdf2image import convert_from_bytes
from PIL import Image
import pytesseract
import re
import io
from googleapiclient.discovery import build
from google.oauth2.service_account import Credentials
from googleapiclient.http import MediaIoBaseDownload, MediaIoBaseUpload

```

```

SCOPE = [
    "https://www.googleapis.com/auth/drive",

```

```

    "https://www.googleapis.com/auth/spreadsheets",
]
CREDENTIALS_FILE = "peer-evaluation-440806-5b8bd496fe1e.json"

# Authenticate Google Drive
def authenticate_drive():
    creds = Credentials.from_service_account_file(CREDENTIALS_FILE,
    scopes=SCOPE)
    service = build('drive', 'v3', credentials=creds)
    return service

# Crop the top left corner of the image
def crop_top_left(image, crop_width, crop_height):
    left = 0
    top = 0
    right = crop_width
    bottom = crop_height
    return image.crop((left, top, right, bottom))

# Get PDF content from Google Drive without downloading
def get_pdf_from_drive(service, file_id):
    request = service.files().get_media(fileId=file_id)
    pdf_content = io.BytesIO()
    downloader = MediaIoBaseDownload(pdf_content, request)

    done = False
    while not done:
        status, done = downloader.next_chunk()
        print(f"Download {int(status.progress() * 100)}%.")

    pdf_content.seek(0) # Move to the beginning of the stream
    return pdf_content

# Extract name from the top-left corner of the first page

```

```

def name_extraction(service, folder_id, file_id, pdf_filename):
    try:
        pdf_content = get_pdf_from_drive(service, file_id)
        images = convert_from_bytes(pdf_content.read())
        #print(f"Successfully converted PDF to images. Number of pages: {len(images)}")
    except Exception as e:
        #print(f"Error converting PDF to images: {e}")
        return False

    if not images:
        raise Exception("Failed to convert PDF to images")
        return False

    image = images[0] # Only the first page is processed
    crop_width = int(image.width * 0.2)
    crop_height = int(image.height * 0.1)

    cropped_image = crop_top_left(image, crop_width, crop_height)

    recognised_text = pytesseract.image_to_string(cropped_image, config='--psm 6')

    extracted_name = re.findall(r'\b\d{3}\b', recognised_text)
    text = "".join(extracted_name)

    if text:
        new_pdf_filename = f"{text}.pdf"
        #print("Extracted name for renaming:", new_pdf_filename)
    else:
        #print("No valid name extracted, keeping the original filename.")
        new_pdf_filename = pdf_filename

    # Re-upload the renamed PDF to Google Drive
    status = upload_pdf(service, folder_id, pdf_content, new_pdf_filename)

```

```

# Delete the original PDF file
status2 = delete_pdf(service, file_id)
if status and status2:
    return True
else:
    return False

# Upload the renamed file back to Google Drive without saving it locally
def upload_pdf(service, folder_id, pdf_content, new_filename):
    pdf_content.seek(0) # Reset the stream position before uploading
    file_metadata = {
        'name': new_filename,
        'parents': [folder_id]
    }
    media = MediaIoBaseUpload(pdf_content, mimetype='application/pdf')
    file = service.files().create(body=file_metadata, media_body=media,
fields='id').execute()
    #print(f"Uploaded renamed file as {new_filename}. File ID: {file.get('id')}")
    return True

# Function to delete the original PDF file
def delete_pdf(service, file_id):
    try:
        service.files().delete(fileId=file_id).execute()
        return True
        #print(f"Deleted original PDF file with ID: {file_id}")
    except Exception as e:
        #print(f"Error deleting file: {e}")
        return False

# Main function to process PDFs in the Google Drive folder
def process_pdfs_in_folder(folder_id):
    count = 0
    service = authenticate_drive()

```

```

# List PDF files in the folder
query = f"'{folder_id}' in parents and mimeType='application/pdf'"
results = service.files().list(q=query, fields="files(id, name)").execute()
files = results.get('files', [])

if not files:
    #raise Exception("No PDF files found in the specified folder")
    return count

# Process each PDF file
for file in files:
    file_id = file['id']
    pdf_filename = file['name']
    #print(f"Processing file: {pdf_filename}")
    status = name_extraction(service, folder_id, file_id, pdf_filename)
    if status:
        count = count + 1

return count

```

**App script files:** - These need to be added to the Google sheets App script section.

### 1. Buttons.gs -

```

function onOpen() {
    var ui = SpreadsheetApp.getUi();
    ui.createMenu('Peer Evaluation')
        .addItem('Pre Evaluation', 'runMainPreEval')
        .addItem('Check Evaluation\'s Pending', 'runCheckEval')
        .addItem('Post Evaluation', 'runMainPostEval')
        .addItem('Send Marks', 'runSendMail')
        .addItem('Send Cumulative Marks', 'runCumulativeMarks')
        .addToUi();
}

```

```

var source_folder = "Pass_Source_Folder_Id";
var target_folder = "Pass_Target_Folder_Id";
var students_per_batch = countStudentsPerBatch();
//var students_per_batch = 2;
var num_Questions;

function countStudentsPerBatch() {
    var folderId = source_folder; // Replace with your folder's ID
    var folder = DriveApp.getFolderById(folderId); // Get the folder by ID
    var files = folder.getFiles(); // Get all files in the folder

    var fileCount = 0;
    var students_per_batch = 0;

    while (files.hasNext()) {
        files.next();
        fileCount++;
    }

    students_per_batch = Math.floor(Math.sqrt(fileCount));

    // Log or return the count of files
    Logger.log('Students per Batch are: ' + students_per_batch);

    return students_per_batch;
}

function doGet(e) {
    // num_Questions = e.parameter.param;
    // // return ContentService.createTextOutput(num_Questions);
    // Logger.log('Number of questions are: ' + num_Questions);
    var action = e.parameter.action; // Get the 'action' parameter from the URL
    if(action == "SetParameter"){

```



```

num_Questions = e.parameter.param;
Logger.log('Number of questions are: ' + num_Questions);
        PropertiesService.getScriptProperties().setProperty("num_Questions",
num_Questions);
    return ContentService.createTextOutput("num_Questions set to: " + num_Questions);
    // return ContentService.createTextOutput(num_Questions);
}
else if (action == "PreEval")
{
    // return ContentService.createTextOutput("hello Rohit " + String(num_Questions));
    return runMainPreEval();
}
else if (action == "CheckEval")
{
    return runCheckEval();
}
else if (action == "PostEval")
{
    return runMainPostEval();
}
else if (action == "GenChart")
{
    return generateCharts();
}
else if (action == "SendMail")
{
    return runSendMail();
}
else if (action == "SendFinalM")
{
    return runCumulativeMarks();
}
else
{

```

```

    return ContentService.createTextOutput("Invalid function call.");
}
}

function runMainPreEval() {
    num_Questions =
    PropertiesService.getScriptProperties().getProperty("num_Questions");
    Logger.log("Running PreEval with num_Questions: " + num_Questions);
    mainPreEval(source_folder, target_folder, students_per_batch, num_Questions);
}

function runCheckEval(){
    //Call the function from Eval Check.gs to check for the peer's who don't evaluated the
    sheets yet
    evalMarksInSheets();
    emailPeerPendingEval();
}

function runMainPostEval() {
    // Calling the mainPreEval to run all the necessary functions
    num_Questions =
    PropertiesService.getScriptProperties().getProperty("num_Questions");
    Logger.log("Running PostEval with num_Questions: " + num_Questions);
    mainPostEval(num_Questions);
}

function runSendMail(){
    //Call the function from Mail.gs to send the final mark's of each student
    sendMailToAllStudents();
}

function runCumulativeMarks(){
    //Call the function from Final Marks.gs to the send the cumulative marks of each
    student

```

```

    sendFinalMarksToAllStudents();
}

```

## 2. PreEvaluation.gs -

```

var sheetName = "PeerEval";

```

```

/**
 * This is the part of the code where
 * Batches are made and then Moved to the Target Folder
 */
function    moveFilesInBatches(source_folder,    target_folder,    students_per_batch,
num_Questions) {
    /**
     * This function moves the files in batches specified by @students_per_batch variable
     * from source folder to target folder
     * identified by @source_folder and @target_folder
     *
     * Name of the sub-folders
     * - The naming is decided by the iteration of the code running
     * - The @folderBatch controls this Script Property
     * - Then a local variable counts each group --> @folderCount
     * - to chnage naming convention, look into these two variables
     *
     * The etire code is written in the try-catch paradigm to make
     * sure any error is handled and shown properly.
     * Any further changes are suggested also to incorporate
     * the same paradigm.
     *
     * To be used in the MAIN Driver Code
     */

    // Load the current value of folderBatch from the Properties Service
    var folderBatch = PropertiesService.getScriptProperties().getProperty('folderBatch');
    if (!folderBatch) {

```

```

    folderBatch = 1; // Initial value if not set previously
}
folderBatch = parseInt(folderBatch); // Convert to integer

try {
    var sourceFolder = DriveApp.getFolderById(source_folder);
} catch (e) {
    Logger.log('Error accessing source folder: ' + e.toString());
    return;
}

try {
    var targetFolder = DriveApp.getFolderById(target_folder);
} catch (e) {
    Logger.log('Error accessing target folder: ' + e.toString());
    return;
}

try {
    var files = sourceFolder.getFiles();
} catch (e) {
    Logger.log('Error retrieving files: ' + e.toString());
    return;
}

var fileCount = 0;
var folderCount = 0;
var currentFolder;

try {
    while (files.hasNext()) {
        if (fileCount % students_per_batch === 0) {
            // Create a new folder for the next batch of files
            var timestamp = new Date().toISOString().replace(/[-:]/g, "");

```

```

        var newFolderName = "G" + folderCount;
        currentFolder = targetFolder.createFolder(newFolderName);
        folderCount++;
    }

    var file = files.next();
    file.makeCopy(currentFolder);
    fileCount++;
}

// Increment folderBatch and save it back to the Properties Service
folderBatch++;

        PropertiesService.getScriptProperties().setProperty('folderBatch',
folderBatch.toString());
    } catch (e) {
        Logger.log('Error moving files: ' + e.toString());
        return;
    }

    // Log the number of files moved and number of folders created
    Logger.log('Moved ' + fileCount + ' files into ' + folderCount + ' folders.');
```

```

}

/**
 * Fetches the names of the folders in which each of the files are located
 * These names are stored in the D-column and are used later
 */
function updateSpreadsheetWithFolderNames(sheetName, target_folder) {
    /**
     * Updates the specified spreadsheet with the folder names based on file names
     *
     * This function processes all files in subfolders of the given target folder,
     * extracts roll numbers from the file names and updates the corresponding row

```

```

* in the spreadsheet with the folder name where the file is located.
*
* @param {string} workbookId - The ID of the Google Sheets workbook.
* @param {string} sheetName - The name of the sheet in the workbook.
* @param {string} target_folder - The ID of the target folder containing subfolders.
*/

```

```

var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
var sheet = spreadsheet.getSheetByName(sheetName);
var targetFolder = DriveApp.getFolderById(target_folder);
var folders = targetFolder.getFolders();

```

```

sheet.getRange(1, 4).setValue("Folder");

```

```

while (folders.hasNext()) {
    var folder = folders.next();
    var folderName = folder.getName();
    var files = folder.getFiles();

```

```

    while (files.hasNext()) {
        var file = files.next();
        var fileName = file.getName();
        Logger.log('Processing file: ' + fileName);

```

```

        // Assume the file name is in the format roll_no.pdf
        var rollNo = extractRollNumberGetName(fileName);

```

```

        if (rollNo !== null) {
            Logger.log('Extracted roll number: ' + rollNo);
            var rowIndex = findRowIndexByValueGetName(sheet, 3, rollNo); // Assuming roll
numbers are in column C (3)

```

```

            if (rowIndex !== -1) {
                sheet.getRange(rowIndex + 1, 4).setValue(folderName); // Update column D (4)
            }

```

```

        Logger.log('Updated roll number ' + rollNo + ' in row ' + (rowIndex + 1) + ' with
folder name ' + folderName);
    } else {
        Logger.log("Roll number " + rollNo + " not found in column C.");
    }
} else {
    Logger.log("Invalid file name: " + fileName);
}
}
}
}
}

```

// Function to extract roll number from file name

```
function extractRollNumberGetName(fileName) {
```

```
    /**
```

```
    * Extracts the roll number from a file name.
```

```
    *
```

```
    * This function assumes the file name format is "roll_no.pdf" and extracts
```

```
    * the numeric roll number from it.
```

```
    *
```

```
    * @param {string} fileName - The name of the file.
```

```
    * @returns {number|null} The extracted roll number, or null if the file name is invalid.
```

```
    * logs the invalid error message
```

```
    */
```

```

        var match = fileName.match(/^(\\d+)\.pdf$/); // Assuming the file name format is
"roll_no.pdf"

```

```
        return match ? parseInt(match[1], 10) : null;
```

```
    }
```

// Function to find the row index by value in a specific column of a sheet

```
function findRowIndexByValueGetName(sheet, columnIndex, value) {
```

```
    /**
```

```
    * Finds the row index by value in a specific column of a sheet.
```

```

*
* This function searches for a specific value in the given column and returns
* the row index where the value is found.
*
* @param {object} sheet - The sheet object where the search is performed.
* @param {number} columnIndex - The column index to search in (1-based).
* @param {number|string} value - The value to search for.
* @returns {number} The row index (0-based) where the value is found, or -1 if not
found.

```

```

*/

var data = sheet.getRange(1, columnIndex, sheet.getLastRow(), 1).getValues();
for (var i = 0; i < data.length; i++) {
    if (data[i][0] === value) {
        return i;
    }
}
return -1; // Value not found
}

```

```

function mainAssign(source_folder, target_folder, students_per_batch, num_Questions) {
    /**
     * Driver code for the Fetching of the Name
     * To be used in the MAIN Driver Code
     */

    updateSpreadsheetWithFolderNames(sheetName, target_folder);
}

```

```

/**
 * Assign the folder which has to be checked by that group of students
 * Make sure the same group is not assigned to them
 */

```



```

function updateBatchAssignments_1(sheetName) {
  /**
   * Updates the batch assignments in a spreadsheet.
   * This function reads folder names from column D, generates a mapping of current
  folders
   * to assigned folders for peer evaluation, and writes the assigned folder names to
  column E.
   *
   * @param {string} workbookId - The ID of the Google Sheets workbook.
   * @param {string} sheetName - The name of the sheet in the workbook.
   */

  var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = spreadsheet.getSheetByName(sheetName);

  // Get folder names from column D, ignoring the first row
  var data = sheet.getRange(2, 4, sheet.getLastRow() - 1, 1).getValues(); // Column D (4)
  var folderNames = data.map(row => row[0]).filter(name => name);

  // Get unique folder names
  var uniqueFolders = Array.from(new Set(folderNames));
  Logger.log('Unique folder names: ' + uniqueFolders.join(', '));

  // Create a mapping of current folder to assigned folder
  var folderMap = {};
  for (var i = 0; i < uniqueFolders.length; i++) {
    var currentFolder = uniqueFolders[i];
    var assignedFolder = uniqueFolders[(i + 1) % uniqueFolders.length];
    folderMap[currentFolder] = assignedFolder;
  }

  Logger.log('Folder assignment map: ' + JSON.stringify(folderMap));

  // Write the assigned folder names to column E, starting from the second row

```

```

for (var i = 0; i < folderNames.length; i++) {
    var folderName = folderNames[i];
    var assignedFolder = folderMap[folderName];

    // Find the row by folder name
    var rowIndex = findRowIndexByValueAssign(sheet, 4, folderName, i + 2); // Start from
row 2
    sheet.getRange(rowIndex, 5).setValue("Assigned Folder");
    if (rowIndex !== -1) {
        sheet.getRange(rowIndex + 1, 5).setValue(assignedFolder); // Update column E (5)
        Logger.log('Assigned ' + assignedFolder + ' to row ' + (rowIndex + 1));
    }
}
}

// Function to find the row index by value in a specific column of a sheet
function findRowIndexByValueAssign(sheet, columnIndex, value, startRow) {
    /**
     * Finds the row index by value in a specific column of a sheet, starting from a specified
row.
     *
     * This function searches for a specific value in the given column, starting from a
specified row,
     * and returns the row index where the value is found.
     *
     * @param {object} sheet - The sheet object where the search is performed.
     * @param {number} columnIndex - The column index to search in (1-based).
     * @param {number|string} value - The value to search for.
     * @param {number} startRow - The row to start the search from (1-based).
     *
     * @returns {number} The row index (0-based) where the value is found, or -1 if not
found.
     */
}

```

```

    var data = sheet.getRange(startRow, columnIndex, sheet.getLastRow() - startRow + 1,
1).getValues();
    for (var i = 0; i < data.length; i++) {
        if (data[i][0] == value) { // Using == to avoid type mismatch
            return i + startRow - 1; // Adjusting the index to be 0-based
        }
    }
    return -1; // Value not found
}

```

```

function    mainAssignBatch(source_folder,    target_folder,    students_per_batch,
num_Questions) {
    /**
     * Driver Code for the assigning of the peer evaluation groups
     * To be used in the MAIN Driver Code
     */

    updateBatchAssignments_1(sheetName);
}

```

```

/**
 * Generate the Link for the assigned Folder for the student to get the
 * VIEW-ONLY Access to the students
 */

function updateBatchAssignments_2(sheetName) {
    /**
     * Updates batch assignments in a spreadsheet for peer evaluation.
     *
     * This function reads folder names from column D, creates a mapping of
     * current folders to assigned folders, and writes the assigned folder
     * names to column E.
     *
     * @param {string} workbookId - The ID of the Google Sheets workbook.

```

```
* @param {string} sheetName - The name of the sheet in the workbook.  
*/
```

```
var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();  
var sheet = spreadsheet.getSheetByName(sheetName);
```

```
// Get folder names from column D, ignoring the first row  
var data = sheet.getRange(2, 4, sheet.getLastRow() - 1, 1).getValues(); // Column D (4)  
var folderNames = data.map(row => row[0]).filter(name => name);
```

```
// Get unique folder names  
var uniqueFolders = Array.from(new Set(folderNames));  
Logger.log('Unique folder names: ' + uniqueFolders.join(', '));
```

```
// Create a mapping of current folder to assigned folder  
var folderMap = {};  
for (var i = 0; i < uniqueFolders.length; i++) {  
  var currentFolder = uniqueFolders[i];  
  var assignedFolder = uniqueFolders[(i + 1) % uniqueFolders.length];  
  folderMap[currentFolder] = assignedFolder;  
}
```

```
Logger.log('Folder assignment map: ' + JSON.stringify(folderMap));
```

```
// Write the assigned folder names to column E, starting from the second row  
for (var i = 0; i < folderNames.length; i++) {  
  var folderName = folderNames[i];  
  var assignedFolder = folderMap[folderName];
```

```
  // Find the row by folder name  
  var rowIndex = findRowIndexByValueLink(sheet, 4, folderName, i + 2); // Start from  
  row 2  
  if (rowIndex !== -1) {  
    sheet.getRange(rowIndex + 1, 5).setValue(assignedFolder); // Update column E (5)
```

```

        Logger.log('Assigned ' + assignedFolder + ' to row ' + (rowIndex + 1));
    }
}

// Function to find the row index by value in a specific column of a sheet
function findRowIndexByValueLink(sheet, columnIndex, value, startRow) {
    /**
     * Finds the row index by value in a specific column of a sheet, starting from a specified
     row.
     *
     * @param {object} sheet - The sheet object where the search is performed.
     * @param {number} columnIndex - The column index to search in (1-based).
     * @param {number|string} value - The value to search for.
     * @param {number} startRow - The row to start the search from (1-based).
     * @returns {number} The row index (0-based) where the value is found, or -1 if not
     found.
     */

    var data = sheet.getRange(startRow, columnIndex, sheet.getLastRow() - startRow + 1,
1).getValues();
    for (var i = 0; i < data.length; i++) {
        if (data[i][0] == value) { // Using == to avoid type mismatch
            return i + startRow - 1; // Adjusting the index to be 0-based
        }
    }
    return -1; // Value not found
}

function generateFolderLinks(sheetName, target_folder) {
    /**
     * Generates view-only links for assigned folders and writes them to the spreadsheet.
     *

```

\* This function reads assigned folders from column E, generates view-only links for these

\* folders, and writes the links to column F. It also sets the sharing settings of the

\* folders to "anyone with the link can view".

\*

\* @param {string} workbookId - The ID of the Google Sheets workbook.

\* @param {string} sheetName - The name of the sheet in the workbook.

\* @param {string} target\_folder - The ID of the target folder containing all folders.

\*/

```
var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
```

```
var sheet = spreadsheet.getSheetByName(sheetName);
```

```
// Get folder assignments from column E, ignoring the first row
```

```
var folderAssignments = sheet.getRange(2, 5, sheet.getLastRow() - 1, 1).getValues(); //
```

```
Column E (5)
```

```
    var uniqueFolders = Array.from(new Set(folderAssignments.map(row =>
row[0]).filter(name => name)));
```

```
// Get target folder
```

```
var targetFolder = DriveApp.getFolderById(target_folder);
```

```
// Create a mapping of folder names to view links
```

```
var folderLinkMap = {};
```

```
var folders = targetFolder.getFolders();
```

```
while (folders.hasNext()) {
```

```
    var folder = folders.next();
```

```
    if (uniqueFolders.includes(folder.getName())) {
```

```
        // Set sharing settings to anyone with the link can view
```

```
                folder.setSharing(DriveApp.Access.ANYONE_WITH_LINK,
DriveApp.Permission.VIEW); // giving VIEW ONLY permission
```

```
var folderId = folder.getId();
```

```
var folderUrl = 'https://drive.google.com/drive/folders/' + folderId;
```

```

        folderLinkMap[folder.getName()] = folderUrl;
    }
}

Logger.log('Folder link map: ' + JSON.stringify(folderLinkMap));

// Write the view-only links to column F, starting from the second row
for (var i = 0; i < folderAssignments.length; i++) {
    var assignedFolder = folderAssignments[i][0];

    if (folderLinkMap[assignedFolder]) {
        var folderLink = folderLinkMap[assignedFolder];

        // Find the row by assigned folder name
        var rowIndex = findRowIndexByValueLink(sheet, 5, assignedFolder, i + 2); // Start
from row 2
        sheet.getRange("E1").setValue("Assigned Folder");
        sheet.getRange("F1").setValue("Assigned Folder Link");
        if (rowIndex !== -1) {
            sheet.getRange(rowIndex + 1, 6).setValue(folderLink); // Update column F (6)
            Logger.log('Assigned link ' + folderLink + ' to row ' + (rowIndex + 1));
        }
    }
}

}

function    mainAssignLink(source_folder,    target_folder,    students_per_batch,
num_Questions) {
    /**
     * Driver code for Link Generation for the viewing of the folders
     * To be part of the MAIN Driver Code
     */

    updateBatchAssignments_2(sheetName);

```

```

    generateFolderLinks(sheetName, target_folder);
}

/**
 * Gets the number of questions and labels the columns
 * Populates the question columns with the question number
 */
function setupQuestionsAndPopulate(num_Questions) {
    /**
     * Setup questions and populate columns.
     */
    Logger.log(num_Questions + '4');
    try {
        var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();
        var sheet = spreadsheet.getSheetByName(sheetName);

        var lastColumn = sheet.getLastColumn();
        Logger.log("LAST COLUMN MEIN DIKKAT" + lastColumn);
        var questionsStartColumns = lastColumn + 1;
        Logger.log("QUESTION START COLUMN MEIN DIKKAT " + questionsStartColumns);
// 7
        Logger.log(num_Questions + '5');
        Logger.log('questionsStartColumns' + questionsStartColumns);
        Logger.log('num_Questions' + num_Questions);
        Logger.log('questionsEndColumns' + (questionsStartColumns + (num_Questions -
1)));
        var questionsEndColumns = (questionsStartColumns + (num_Questions - 1));
        Logger.log("QUESTION END COLUMN MEIN DIKKAT" + questionsEndColumns); //
73
        Logger.log(questionsEndColumns); //73 on putting 4
        for (var i = questionsStartColumns; i <= questionsEndColumns; i++) {
            sheet.getRange(1, i).setValue('Question ' + (i - questionsStartColumns + 1));
        }
    }
}

```



```

var rollNumbers = sheet.getRange(2, 4, sheet.getLastRow() - 1, 1).getValues();

for (var j = 0; j < rollNumbers.length; j++) {
    for (var k = questionsStartColumns; k <= questionsEndColumns; k++) {
        sheet.getRange(j + 2, k).setValue('Q' + (k - questionsStartColumns + 1)); // Write
question numbers under each question label
    }
}

Logger.log(num_Questions + '6');
    Logger.log('Successfully set up ' + num_Questions + ' questions and populated the
columns with question numbers.');
```

```

    } catch (e) {
        Logger.log('Error in setting up questions and populating columns: ' + e.toString());
    }
}

// function vaibhav() {
//     setupQuestionsAndPopulate(4);
// }

function mainSetupQuestions(num_Questions) {
    /**
     * Driver code for writing the number of questions
     * To be a part of the MAIN Driver Code
     */
    Logger.log(num_Questions + '3');
    setupQuestionsAndPopulate(num_Questions);
}

/**
 * Making the new spreadsheet for each student
 * Tracking the link for those sheets in the Last Column of the sheet

```

\* The sheet is made in the target folder identified by @target\_folder

\*

\* Sheet Protection and access is properly set up

\* Only the intended cells can be edited by the user

\*/

```
function createStudentWorkbooks(source_folder, target_folder, students_per_batch,  
num_Questions) {
```

```
  /**
```

\* The code iterates through the Unique IDs and makes workbooks for them to

\* evaluate the peers.

\*

\* Once it runs, it makes the workbooks in the target folder identified by

\* @target\_folder in the global variables.

\*

\* It puts the links of the workbooks for the respective workbook in the last

\* column of the sheet identifies by the @sheetName variable

\*

\* It then gives access only to the respective student and nobody else.

\*

\* To be a part of the MAIN Driver Code

```
  */
```

```
  var mainWorkbook = SpreadsheetApp.getActiveSpreadsheet();
```

```
  var sheet = mainWorkbook.getSheetByName(sheetName);
```

```
  if (!sheet) {
```

```
    Logger.log('Sheet not found.');
```

```
    return;
```

```
  }
```

```
  var data = sheet.getDataRange().getValues();
```

```
  var folder = DriveApp.getFolderById(target_folder);
```

```
  sheet.getRange(1, data[0].length + 1).setValue("Spreadsheet Link");
```

```

for (var i = 1; i < data.length; i++) { // Start from 1 to skip the header row
    var name = data[i][0]; // Column A
    var email = data[i][1]; // Column B
    var uniqueID = data[i][2]; // Column C
    var folderName = data[i][4]; // Column E

    Logger.log(uniqueID);
    // Create a new workbook for the student
    var studentWorkbook = SpreadsheetApp.create(name);
    // Apply sharing permissions
    var studentFile = DriveApp.getFileById(studentWorkbook.getId());
        studentFile.setSharing(DriveApp.Access.ANYONE_WITH_LINK,
DriveApp.Permission.EDIT);

    var studentSheet = studentWorkbook.getActiveSheet();

    // Write the name and unique ID
    studentSheet.getRange('A1').setValue(name);
    studentSheet.getRange('A2').setValue(uniqueID);

    // Write the question numbers
    for (var q = 1; q <= num_Questions; q++) {
        studentSheet.getRange(2, q + 1).setValue('Question ' + q);
    }

    // Get all unique IDs in column D that have the specified folder name in column E
    var uniqueIDsWithFolder = [];
    for (var j = 1; j < data.length; j++) {
        if (data[j][3] == folderName) {
            uniqueIDsWithFolder.push(data[j][2]); // Column C
        }
    }
}

```

```

// Write these unique IDs in column A of the student sheet
for (var k = 0; k < uniqueIDsWithFolder.length; k++) {
    studentSheet.getRange(k + 3, 1).setValue(uniqueIDsWithFolder[k]);
}

// Protect the student sheet
// var protection = studentSheet.protect().setDescription('Protected Sheet');
// protection.removeEditors(protection.getEditors()); // Remove all editors
// protection.addEditor(email);

// Protect specific ranges (Column A and rows 1 and 2)
protectRange(studentSheet.getRange('A:A'));
protectRange(studentSheet.getRange('1:1'));
protectRange(studentSheet.getRange('2:2'));
var lastRow = studentSheet.getLastRow();
var nextRow = lastRow + 1;
protectRange(studentSheet.getRange(nextRow + ':' + nextRow));

var lastColumn = studentSheet.getLastColumn();
var nextColumn = lastColumn + 1;
protectRange(studentSheet.getRange(nextColumn + ':' + nextColumn));

// Move the student workbook to the specified folder
var studentFile = DriveApp.getFileById(studentWorkbook.getId());
folder.addFile(studentFile);
DriveApp.getRootFolder().removeFile(studentFile);

// Generate the edit link for the student workbook
var editLink = studentWorkbook.getUrl();

// Add the edit link to the main sheet in the corresponding row and last column

```

```
        sheet.getRange(i + 1, data[0].length + 1).setValue(editLink); // Assuming data[0]
contains headers
```

```
        // Share the student workbook with the student email
        // studentWorkbook.addEditor(email);
            // studentWorkbook.setSharing(DriveApp.Access.ANYONE_WITH_LINK,
DriveApp.Permission.EDIT);
    }
}
```

```
    Logger.log('Student workbooks created and permissions assigned.');
```

```
}
```

```
function protectRange(range) {
```

```
    /**
```

```
    * This function is made to protect the cells
```

```
    * Used by createStudentWorkbooks() to protect
```

```
    * the sheets before sending them to the students
```

```
    *
```

```
    * @param {string} range - The range of the cells that are to be protected
```

```
    * Prevents the editing of the cells.
```

```
    */
```

```
    var protection = range.protect().setDescription('Protected Range');
```

```
    protection.removeEditors(protection.getEditors()); // Remove all editors
```

```
    protection.setWarningOnly(false); // Prevent editing
```

```
}
```

```
/**
```

```
    * The Final Driver Code MAIN
```

```
    * All Functionalities are to be called from this code
```

```
    */
```

```

function mainPreEval(source_folder, target_folder, students_per_batch, num_Questions)
{

    Logger.log('Starting mainProcess...');
    Logger.log(num_Questions + '1');

    //Moving the files to the target folder
    try {
        Logger.log('Moving files in batches to the target folder...');
        moveFilesInBatches(source_folder, target_folder, students_per_batch,
num_Questions);
        Logger.log('Files moved successfully.');
```

```

    } catch (e) {
        Logger.log('Error moving files: ' + e.toString());
        return;
    }

    // Mapping the students to their folders
    try {
        Logger.log('Updating spreadsheet with folder names...');
        mainAssign(source_folder, target_folder, students_per_batch, num_Questions);
        Logger.log('Spreadsheet updated with folder names successfully.');
```

```

    } catch (e) {
        Logger.log('Error updating spreadsheet with folder names: ' + e.toString());
        return;
    }

    // Assigning the folder for the students to evaluate
    try {
        Logger.log('Assigning peer evaluation groups...');
        mainAssignBatch(source_folder, target_folder, students_per_batch, num_Questions);
        Logger.log('Peer evaluation groups assigned successfully.');
```

```

    } catch (e) {
        Logger.log('Error assigning peer evaluation groups: ' + e.toString());
    }
}

```

```

        return;
    }

    // Genrating view-only links for the folder for the student evaluation
    try {
        Logger.log('Generating view-only links for assigned folders...');
        mainAssignLink(source_folder, target_folder, students_per_batch, num_Questions);
        Logger.log('View-only links generated successfully.');
```

```

    } catch (e) {
        Logger.log('Error generating view-only links: ' + e.toString());
        return;
    }

    // Question labelling in the Main Sheet
    try {
        Logger.log("Starting to Setup the question labelling");
        Logger.log(num_Questions + '2');
        mainSetupQuestions(num_Questions);
        Logger.log("Successfully labelled the questions for each student");
    } catch (e) {
        Logger.log("Error in labelling the question columns. Error Code: " + e.toString());
        return;
    }

    // Making the workbook for student evaluation
    try {
        Logger.log("Starting to make the workbooks, making links and writing them in the
sheet");
        createStudentWorkbooks(source_folder, target_folder, students_per_batch,
num_Questions);
        Logger.log("Completed making workbooks and tracking links successfully!");
    } catch (e) {
        Logger.log("Error in making the workbooks for evaluation. Error Code: " +
e.toString());
    }

```

```

    return;
}

Logger.log('mainProcess completed successfully.');
```

```

}

function driverCheck(){
    var source = '1h_Cd93RHXYMvjF2L5G5D70z7x25r6yqb';
    var target = '14Uu6G4frYSj9dWcE7Ww28NoBDz2dyTPU';
    var student = 2;
    var questions = 5;
    mainPreEval(source, target, student, questions);
}

```

### 3. Eval Check.gs -

```

function evalMarksInSheets() {

    var mainSheetName = "PeerEval";

    var mainSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(mainSheetName);

    var headers = mainSheet.getRange(1, 1, 1, mainSheet.getLastColumn()).getValues()[0];

    var linksColIndex = headers.indexOf("Spreadsheet Link");
    if (linksColIndex === -1) {
        Logger.log('Spreadsheet Link column not found.');
```

```

    return;
}

linksColIndex += 1;

var evaluationColIndex = headers.indexOf("Evaluation");

```



```

if (evaluationCollIndex === -1) {
    evaluationCollIndex = headers.length;
    mainSheet.getRange(1, evaluationCollIndex + 1).setValue("Evaluation");
} /*else {
    evaluationCollIndex += 1;
}*/

    var data = mainSheet.getRange(2, linksCollIndex, mainSheet.getLastRow() - 1,
1).getValues();

for (var i = 0; i < data.length; i++) {
    var sheetLink = data[i][0];

    if (sheetLink) {
        try {
            var spreadsheet = SpreadsheetApp.openByUrl(sheetLink);

            var sheetToCheck = spreadsheet.getActiveSheet();
            var columnToCheck = 2;

            var range = sheetToCheck.getRange(1, columnToCheck,
sheetToCheck.getLastRow()).getValues();

            var marksFound = false;
            for (var j = 0; j < range.length; j++) {
                if (typeof range[j][0] === 'number' && !isNaN(range[j][0])) {
                    marksFound = true;
                    break;
                }
            }

            if (marksFound) {
                mainSheet.getRange(i + 2, evaluationCollIndex+1).setValue("Done");
            } else {
                mainSheet.getRange(i + 2, evaluationCollIndex+1).setValue("Not done");
            }
        }
    }
}

```

```

    }

    } catch (e) {
        Logger.log("Error opening sheet: " + sheetLink);
        mainSheet.getRange(i + 2, evaluationColIndex).setValue("Error Accessing Sheet");
    }
    } else {
        mainSheet.getRange(i + 2, evaluationColIndex).setValue("No Link Provided");
    }
    }
}

```

```

function emailPeerPendingEval() {

```

```

    var mainSheetName = "PeerEval";

```

```

        var mainSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(mainSheetName);

```

```

        var headers = mainSheet.getRange(1, 1, 1, mainSheet.getLastColumn()).getValues()[0];

```

```

        var nameColIndex = headers.indexOf("Name");

```

```

        var emailColIndex = headers.indexOf("EMail ID");

```

```

        var evaluationColIndex = headers.indexOf("Evaluation");

```

```

        if (nameColIndex === -1 || emailColIndex === -1 || evaluationColIndex === -1) {

```

```

            Logger.log("Required columns not found.");

```

```

            return;

```

```

        }

```

```

        nameColIndex += 1;

```

```

        emailColIndex += 1;

```

```

evaluationCollIndex += 1;

    var data = mainSheet.getRange(2, 1, mainSheet.getLastRow() - 1,
mainSheet.getLastColumn()).getValues();

for (var i = 0; i < data.length; i++) {
    var name = data[i][nameCollIndex - 1];
    var email = data[i][emailCollIndex - 1];
    var evaluationStatus = data[i][evaluationCollIndex - 1];

    if (evaluationStatus === "Not done" && email) {

        var subject = "Gentle Reminder! For Pending Evaluation";

        var body = "Dear " + name + ",<br><br>" +
            "Our records indicate that you have not yet completed your evaluation. " +
            "Please complete it as soon as possible.<br><br>" +
            "Best regards,<br>CSE, IIT Ropar";

        MailApp.sendEmail({
            to: email,
            subject: subject,
            htmlBody: body
        });

        Logger.log('Email sent to: ' + email);
    }
}
}

```

#### 4. PostEvaluation.gs -

```
/**
 * GLOBAL VARIABLE
 *
 * The Global Variables are made for common use and these are the
 * ones that will be needed to change when running on a different platform
 */

/**
 * The sheet where all the results for all students are consolidated
 * as per their unique IDs. The sheet is made on its own if not
 * made already. Just run the code accordingly.
 */
var consolidationSheetName = "Consolidation Results";

/**
 * The evaluation sheet sets up the analytics of the results
 * Also is the final evaluation consolidation
 */
var evaluationSheetName = 'Evaluation Results'; // Name for the results sheet

function peerReviewConsolidation(num_Questions) {
  /**
   * Consolidates peer review data from multiple workbooks into a single sheet.
   *
   * This function performs the following tasks:
   * 1. Opens the specified source and target sheets.
   * 2. Clears the target sheet if it already exists, or creates a new one if it doesn't.
   * 3. Copies specific columns (first and third) from the source sheet to the target sheet.
   * 4. Reads unique IDs from the copied data and creates a dictionary mapping these
   IDs to their row numbers.
   * 5. Iterates through each link in the source sheet, opening each linked workbook.
```

\* 6. Extracts specified data from each linked workbook and inserts it into the target sheet.

\* 7. Updates the dictionary to keep track of the new row positions as data is inserted.

\* 8. Adds headers to the target sheet.

\*

\* @param {string} workbookId - The ID of the main workbook.

\* @param {string} sheetName - The name of the source sheet containing peer review links and data.

\* @param {string} consolidationSheetName - The name of the target sheet where consolidated data will be stored.

\* @param {number} num\_Questions - The number of questions for which data is collected.

\*/

```
var workbook = SpreadsheetApp.getActiveSpreadsheet();
var sourceSheet = workbook.getSheetByName(sheetName);
var targetSheet = workbook.getSheetByName(consolidationSheetName);
```

```
if (!targetSheet) {
    targetSheet = workbook.insertSheet(consolidationSheetName);
} else {
    targetSheet.clear(); // Clear existing content
}
```

```
var sourceSheetvalues = sourceSheet.getDataRange().getValues();
```

```
var students_count = sourceSheet.getLastRow() - 1;
```

```
// Extract the first and third columns from the source sheet
```

```
var dataToCopy = [];
```

```
for (var i = 1; i < sourceSheetvalues.length; i++) { // start from 1 to skip the header row
```

```
    var row = [];
```

```
    row.push(sourceSheetvalues[i][0]); // first column
```

```
    row.push(sourceSheetvalues[i][2]); // third column
```

```

    dataToCopy.push(row);
}

// Write the extracted data to the target sheet
    targetSheet.getRange(2, 1, dataToCopy.length,
dataToCopy[0].length).setValues(dataToCopy); // start from row 2 to skip the header row

// Read the values from the target sheet
var targetSheetValues = targetSheet.getDataRange().getValues();

// Create a dictionary with keys as values from the second column and values as their
respective row numbers
var dict = {};
for (var i = 1; i < targetSheetValues.length; i++) { // start from 1 to skip the header row
    var key = targetSheetValues[i][1]; // second column
    dict[key] = i + 1; // store the row number (i + 1 because array is 0-based but sheet is
1-based)
}

// Open the workbook and loop through all rows from the 3rd row
for (var i = 1; i < sourceSheetvalues.length; i++) {
    Logger.log(num_Questions);
    var sourceWorkbookLink = sourceSheetvalues[i][+num_Questions + 6];
    Logger.log(num_Questions + 6); // 46
    Logger.log(sourceWorkbookLink);
    var sourceWorkbook = SpreadsheetApp.openByUrl(sourceWorkbookLink);
    var sourceWorkbookSheet = sourceWorkbook.getSheets()[0]; // assuming we want
the first sheet
    var sourceWorkbookValues = sourceWorkbookSheet.getDataRange().getValues();

    // Get the values of the first column
    var values = sourceWorkbookSheet.getRange('A:A').getValues();

    for (var j = 2; j < sourceWorkbookValues.length; j++) { // start from row 3

```

```

// for (var j = 2; j < k+1 ; j++) { // start from row 3
  var key = sourceWorkbookValues[j][0]; // first column
  Logger.log(key)
  var value = dict[key];
  if (value !== undefined) {
    Logger.log(value);

    // var lastColumn = sourceWorkbookSheet.getLastColumn();

    // Extract the row except the first column
    var rowToInsert = sourceWorkbookValues[j].slice(1, num_Questions+1);

    var sum = 0;
    for (var m = 0; m < rowToInsert.length; m++) {
      sum += rowToInsert[m];
    }
    rowToInsert.push(sum)

    Logger.log(rowToInsert)

    // Insert the extracted row into the target sheet at the specified position
    targetSheet.insertRowAfter(value); // Insert a row before the specified row number
    targetSheet.getRange(value, 3, 1, rowToInsert.length).setValues([rowToInsert]); //
    Insert values into the new row

    dict[key]++;

    // Update values of all subsequent keys in the dictionary
    for (var updateKey in dict) {
      if (updateKey > key) {
        dict[updateKey]++;
      }
    }
  }
}

```

```

    }
}

// Add headers to the target sheet
var headers = ["Name", "Unique ID"];
for (var i = 1; i <= num_Questions; i++) {
    headers.push("Q" + i);
}
headers.push("Total");

targetSheet.getRange(1, 1, 1, headers.length).setValues([headers]);

}

function processEvaluationData() {
    /**
     * Takes the total values given in the consolidation sheet
     * and puts it in a column format in the Evaluation Sheet
     *
     * The algorithm can accomodate different number of peers
     * per student.
     */

    var workbook = SpreadsheetApp.getActiveSpreadsheet();
    var finalSheet = workbook.getSheetByName(consolidationSheetName);
    var resultsSheet = workbook.getSheetByName(evaluationSheetName);

    // Create results sheet if it doesn't exist
    if (!resultsSheet) {
        resultsSheet = workbook.insertSheet(evaluationSheetName);
    } else {
        resultsSheet.clear(); // Clear existing content
    }
}

```



```

var finalData = finalSheet.getDataRange().getValues();
var numRows = finalSheet.getLastRow();
var numCols = finalSheet.getLastColumn();

// Set up headers for the results sheet
var headers = ['Peer'];

// Collect unique IDs and set as headers in results sheet
for (var row = 1; row < numRows; row++) {
    var uniqueID = finalSheet.getRange(row + 1, 2).getValue(); // Assuming unique ID is in
column B (second column)
    if (uniqueID && !headers.includes(uniqueID)) {
        headers.push(uniqueID);
    }
}
resultsSheet.getRange(1, 1, 1, headers.length).setValues([headers]);

// Extract marks for each unique ID and place them under respective columns
for (var row = 2; row <= numRows; row++) {
    var uniqueID = finalSheet.getRange(row, 2).getValue(); // Assuming unique IDs are in
column B
    if (uniqueID && headers.includes(uniqueID)) {
        var lastCol = finalSheet.getLastColumn();
        var marks = [];
        for (var i = row; i <= numRows; i++) {
            var mark = finalSheet.getRange(i, lastCol).getValue();
            if (mark === "") break;
            marks.push(mark);
        }
        // Place marks under respective unique ID in results sheet
        var colIndex = headers.indexOf(uniqueID) + 1;
        for (var i = 0; i < marks.length; i++) {
            resultsSheet.getRange(i + 2, colIndex).setValue(marks[i]);
        }
    }
}

```

```
}  
}
```

```
Logger.log("Evaluation data processed and copied to Evaluation Results sheet.");
```

```
try {  
    Logger.log("Starting to put labels in the A column");  
    putLabel();  
    Logger.log("Lables put successfully!");  
} catch (e) {  
    Logger.log("Error in labelling. Error Code: " + e.toString());  
    return;  
}
```

```
try {  
    Logger.log("Starting to put colors in the labels in the A column");  
    putColor();  
    Logger.log("Colors put successfully!");  
} catch (e) {  
    Logger.log("Error in coloring. Error Code: " + e.toString());  
    return;  
}  
}
```

```
function putLabel() {  
    /**  
    * This function is used to put Labels for the Column A in the Evaluation Results sheet  
    * Also the marks given by the TA and some of the metrics to be used for visualizations  
    */
```

```
var workbook = SpreadsheetApp.getActiveSpreadsheet();  
var resultsSheet = workbook.getSheetByName(evaluationSheetName);
```

```
var lastRow = resultsSheet.getLastRow();
```

```

// Loop from row 2 to the last row and set the label in column A
for (var i = 2; i <= lastRow; i++) {
    var label = 'Peer ' + (i - 1); // Create label "peer 1", "peer 2", etc.
    resultsSheet.getRange(i, 1).setValue(label); // Set the label in column A
}

resultsSheet.getRange(lastRow + 1, 1).setValue("TA");
resultsSheet.getRange(lastRow + 2, 1).setValue("SD");
resultsSheet.getRange(lastRow + 3, 1).setValue("Peer Average");
resultsSheet.getRange(lastRow + 4, 1).setValue("SD of SD");
}

function putColor() {
    /**
     * Puts color to the labels
     */

    var workbook = SpreadsheetApp.getActiveSpreadsheet();
    var resultsSheet = workbook.getSheetByName(evaluationSheetName);

    var lastRow = resultsSheet.getLastRow();
    var lastColumn = resultsSheet.getLastColumn();

    // Define the color
    var color = '#FFDDC1'; // Light orange color

    // Color the first row till the last column
    resultsSheet.getRange(1, 1, 1, lastColumn).setBackground(color);

    // Color the first column till the last row
    resultsSheet.getRange(1, 1, lastRow, 1).setBackground(color);
}

```

```

function calculateStatistics() {
  /**
   * Makes the necessary calculations for the Average and standard deviation
   */

  var workbook = SpreadsheetApp.getActiveSpreadsheet();
  var resultsSheet = workbook.getSheetByName(evaluationSheetName);

  var lastRow = resultsSheet.getLastRow();
  var lastColumn = resultsSheet.getLastColumn();

  // Find the row with 'SD' in column A
  var sdRow = -1;
  for (var row = 1; row <= lastRow; row++) {
    var cellValue = resultsSheet.getRange(row, 1).getValue();
    if (cellValue === 'SD') {
      sdRow = row;
      break;
    }
  }

  if (sdRow === -1) {
    Logger.log("SD label not found in column A.");
    return;
  }

  // Calculate Peer Average
  var peerAverageRange = resultsSheet.getRange(2, 2, lastRow - 1, lastColumn - 1);
  var peerAverage = [];

  for (var col = 2; col <= lastColumn; col++) {
    var columnData = resultsSheet.getRange(2, col, lastRow - 1).getValues();
    var validData = columnData.filter(function(row) { return row[0] !== "";
  }).map(function(row) { return row[0]; });

```

```

    var sum = validData.reduce(function(acc, value) { return acc + value; }, 0);
    var average = validData.length ? sum / validData.length : 0;
    peerAverage.push(average);
  }
  resultsSheet.getRange(sdRow + 1, 2, 1, lastColumn - 1).setValues([peerAverage]);

  // Calculate SD
  var sdArray = [];

  for (var col = 2; col <= lastColumn; col++) {
    var columnData = resultsSheet.getRange(2, col, lastRow - 1).getValues();
    var validData = columnData.filter(function(row) { return row[0] !== "";
  }).map(function(row) { return row[0]; });
    var mean = validData.reduce(function(acc, value) { return acc + value; }, 0) /
    validData.length;
    var sd = Math.sqrt(validData.reduce(function(acc, value) { return acc +
    Math.pow(value - mean, 2); }, 0) / validData.length);
    sdArray.push(sd);
  }
  resultsSheet.getRange(sdRow, 2, 1, lastColumn - 1).setValues([sdArray]);

  // Calculate SD of SD
  var sdOfSdMean = sdArray.reduce(function(acc, value) { return acc + value; }, 0) /
  sdArray.length;
  var sdOfSd = Math.sqrt(sdArray.reduce(function(acc, value) { return acc +
  Math.pow(value - sdOfSdMean, 2); }, 0) / sdArray.length);
  resultsSheet.getRange(sdRow + 2, 2).setValue(sdOfSd);

  // Add labels in column A
  resultsSheet.getRange(sdRow + 1, 1).setValue('Peer Average');
  resultsSheet.getRange(sdRow + 2, 1).setValue('SD of SD');

  Logger.log('Statistics calculated and added to the Evaluation Results sheet.');
```

```

/**
 * Main Code to be run finally
 */
function mainPostEval(num_Questions) {
  /**
   * The main driver function for the Peer Evaluation Automation process.
   * This function will sequentially call all necessary steps:
   * - Rename files in the source folder
   * - Move files in batches to the target folder
   * - Update the spreadsheet with folder names
   * - Assign peer evaluation groups
   * - Generate view-only links for the assigned folders
   *
   * Each step is wrapped in a try-catch block to handle and log errors.
   *
   * @changelog
   *
   * version beta
   * - 19-06-2024
   * - only till the link generation
   *
   * version 1.1
   * - 21-06-2024 Summer Solstice
   * - complete system made.
   * - Sends the mails for the evaluation
   * - Next part is the consolidation of evaluation
   */

```

```

  Logger.log('Starting mainPostEval...');

```

```

  // Consolidating the reviews in one place, the Consolidation Sheet

```

```

  try {

```

```

    Logger.log('Peer Review Consolidation starting...');

```

```

    peerReviewConsolidation(num_Questions);
    Logger.log('Peer Review Consolidated.');
```

} catch (e) {

```

    Logger.log('Error consolidating reviews: ' + e.toString());
    return;
}
```

// Process the peer evaluation data

```

try {
    Logger.log('Data Evaluation starts...');
    processEvaluationData();
    Logger.log('Data evaluated.');
```

} catch (e) {

```

    Logger.log('Error in evaluating data: ' + e.toString());
    return;
}
```

// Consolidating the reviews in one place, the Consolidation Sheet

```

try {
    Logger.log('Updating statistics...');
    calculateStatistics();
    Logger.log('Statistics Updated.');
```

} catch (e) {

```

    Logger.log('Error in calculating statistics: ' + e.toString());
    return;
}
```

```

    Logger.log('mainPostEval completed successfully.');
```

}

## 5. Mail.gs -

```
function sendMailToAllStudents() {  
    mapPeerAverageMarks()  
    mapCumulativeMarks()  
}
```

```
function mapPeerAverageMarks() {
```

```
    var sourceSheetName = "Evaluation Results";
```

```
    var targetSheetName = "PeerEval";
```

```
    var sourceSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(sourceSheetName);  
    var targetSheet = SpreadsheetApp.getActiveSpreadsheet().getSheetByName(targetSheetName);
```

```
    var sourceData = sourceSheet.getDataRange().getValues();
```

```
    var peerAverageRow = -1;
```

```
    for (var i = 0; i < sourceData.length; i++) {  
        if (sourceData[i][0] === "Peer Average") {  
            peerAverageRow = i;  
            break;  
        }  
    }  
}
```

```
    if (peerAverageRow === -1) {  
        Logger.log('Peer Average row not found.');
```

```
        return;
```

```
    }
```

```
    var peerIDs = sourceData[0];
```

```
    var peerAverageMarks = sourceData[peerAverageRow];
```



```

var targetData = targetSheet.getDataRange().getValues();

var headers = targetData[0];
var averageMarksColIndex = headers.indexOf("Average Marks");

if (averageMarksColIndex === -1) {
    averageMarksColIndex = headers.length;
    targetSheet.getRange(1, averageMarksColIndex + 1).setValue("Average Marks");
}

for (var i = 1; i < targetData.length; i++) {
    var targetPeerID = targetData[i][2];
    var peerIndex = peerIDs.indexOf(targetPeerID);

    if (peerIndex !== -1) {
        var averageMark = peerAverageMarks[peerIndex];
        targetSheet.getRange(i + 1, averageMarksColIndex + 1).setValue(averageMark);
    } else {
        targetSheet.getRange(i + 1, averageMarksColIndex + 1).setValue("Not Found");
    }
}
}

```

## 6. Final Marks.gs -

```

function sendFinalMarksToAllStudents() {
    //mapCumulativeMarks()
    sendMarksByEmail()
}

function mapCumulativeMarks() {
    var sourceSheetName = "PeerEval"; // Source worksheet name
    var targetSheetName = "Cumulative Marks"; // Target worksheet name (to be created if
    not present)
}

```

```

var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

// Get the source sheet
var sourceSheet = spreadsheet.getSheetByName(sourceSheetName);
if (!sourceSheet) {
    Logger.log("Source sheet not found: " + sourceSheetName);
    return;
}

// Check if the target sheet exists, if not, create it
var targetSheet = spreadsheet.getSheetByName(targetSheetName);
if (!targetSheet) {
    Logger.log("Target sheet not found. Creating a new one: " + targetSheetName);
    targetSheet = spreadsheet.insertSheet(targetSheetName);
}

// Get data from the source sheet
var sourceData = sourceSheet.getDataRange().getValues();

// Find the "EMail ID" column in the source sheet
var sourceHeaders = sourceData[0]; // First row of the source sheet
var emailIdColIndexSource = sourceHeaders.indexOf("EMail ID");

if (emailIdColIndexSource === -1) {
    Logger.log('EMail ID column not found in the source sheet.');
```

return;

```

}

// Find the "Average Marks" column in the source sheet
var averageMarksColIndexSource = sourceHeaders.indexOf("Average Marks");

if (averageMarksColIndexSource === -1) {
    Logger.log("Average Marks column not found in the source sheet.");

```

```

    return;
}

// Retrieve the relevant columns for Peer Emails and Peer Average Marks from the
source sheet
var peerEmails = sourceData.slice(1).map(row => row[emailIdColIndexSource]); // Get
email IDs from the column
var peerAverageMarks = sourceData.slice(1).map(row =>
row[averageMarksColIndexSource]); // Get Average Marks from the column

// Get the data from the target sheet (Cumulative Marks)
var targetData = targetSheet.getDataRange().getValues();

// Check if "Email_Id" column exists in the target sheet, if not, create it
var targetHeaders = targetData.length ? targetData[0] : [];
var emailIdColIndexTarget = targetHeaders.indexOf("Email_Id");

if (emailIdColIndexTarget === -1) {
    Logger.log("Email_Id column not found in the target sheet. Adding it.");

    // If "Email_Id" column is not found, add it as the first column
    emailIdColIndexTarget = 0;
    targetSheet.insertColumnBefore(1);
    targetSheet.getRange(1, 1).setValue("Email_Id");
    targetHeaders = ["Email_Id"].concat(targetHeaders); // Update the header
}

// Add a new column for Average Marks with a unique label
var timestamp = new Date().toLocaleString(); // Get a timestamp to create a unique
column header
var newColHeader = "Marks_" + timestamp;
var newColIndex = targetHeaders.length + 1;
targetSheet.getRange(1, newColIndex).setValue(newColHeader); // Add the new
header in the target sheet

```

```

// Iterate over the peerEmails from the source sheet and map the marks
for (var i = 0; i < peerEmails.length; i++) {
    var peerEmail = peerEmails[i];
    if (!peerEmail) continue; // Skip if there's no email ID in the source sheet

    var peerAverage = peerAverageMarks[i];
    var foundInTarget = false;

    // Search for the email ID in the target sheet
    for (var j = 1; j < targetData.length; j++) {
        var targetEmailId = targetData[j][emailIdColIndexTarget];
        if (targetEmailId === peerEmail) {
            targetSheet.getRange(j + 1, newColIndex).setValue(peerAverage); // Update marks
            if email found
                foundInTarget = true;
                break;
        }
    }

    // If the email ID was not found in the target sheet, append it at the end
    if (!foundInTarget) {
        var newRowIndex = targetSheet.getLastRow() + 1;
        targetSheet.getRange(newRowIndex, emailIdColIndexTarget +
1).setValue(peerEmail); // Add new email
        targetSheet.getRange(newRowIndex, newColIndex).setValue(peerAverage); // Add
corresponding average mark
    }
}

Logger.log("Marks mapping completed with a new column added: " + newColHeader);
}

```

```

function sendMarksByEmail() {
    var targetSheetName = "Cumulative Marks"; // Target sheet name
    var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

    // Get the target sheet
    var targetSheet = spreadsheet.getSheetByName(targetSheetName);
    if (!targetSheet) {
        Logger.log("Target sheet not found: " + targetSheetName);
        return;
    }

    // Get the target sheet data
    var targetData = targetSheet.getDataRange().getValues();
    if (targetData.length === 0) {
        Logger.log("Target sheet is empty.");
        return;
    }

    // Get the headers (first row) of the target sheet
    var headers = targetData[0];

    // Find the "Email_Id" column index
    var emailIdColIndex = headers.indexOf("Email_Id");
    if (emailIdColIndex === -1) {
        Logger.log("Email_Id column not found in the target sheet.");
        return;
    }

    // Find the columns with "Marks_timestamp" format
    var marksColumns = [];
    for (var i = 0; i < headers.length; i++) {
        if (headers[i].startsWith("Marks_")) {
            marksColumns.push(i);
        }
    }
}

```

```

    }
}

if (marksColumns.length === 0) {
    Logger.log("No Marks_timestamp columns found in the target sheet.");
    return;
}

// Loop through each row and calculate total marks, then send an email
for (var i = 1; i < targetData.length; i++) {
    var emailId = targetData[i][emailIdColIndex];

    if (emailId) {
        var totalMarks = 0;

        // Sum all the marks from the "Marks_timestamp" columns
        for (var j = 0; j < marksColumns.length; j++) {
            var mark = parseFloat(targetData[i][marksColumns[j]]);
            if (!isNaN(mark)) {
                totalMarks += mark;
            }
        }

        // Send an email with the total marks
        var subject = "Your Total Marks in the Course";
        var body = "Dear Student,\n\nYour total marks in the course are: - " + totalMarks +
            "\n\nThanks & Regards,\nCSE, IIT Ropar";

        MailApp.sendEmail(emailId, subject, body);
        Logger.log("Email sent to: " + emailId + " with total marks: " + totalMarks);
    }
}
}

```