Code Base for the Peer Evaluation Automation System (PEAS)

Pre - Evaluation Google App Script

/\*\*

\* @name PEER EVALUATION AUTOMATION (Pre-Evaluation System)

\*

\* @fileoverview The code is made to accomodate the Peer Evaluation System throughout the G-suit.

\* will include the Googel Drive and Google Sheets

\*

\*

\* @version 1.1

\*

\* @changelog

\* - version beta

\* - released 19-06-2024

\* -Issues

\* - change the folder name to just G1 G2 G3

\* - Issues Resolved on 20-06-2024

\*

\* - version 1.0

\* - released 20-06-2024

\* - Link creation works but the problem is with access and permissions

\* - The sheets were being created inside the main workbook

\* - This lead to many problems

\* - Approach abandoned in the evening of 20-06-2024

\*

\* - version 1.1

\* - Spreadsheets are being made in a seperate folder (Target Folder)

\* - The links are being tracked and the permissions are being set properly

\* - The mail of the links is being sent properly and internal testing is being done

\* - Released 21-06-2024

\*

\*/

/\*\*

\* BIG NOTE:

\*

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\*

\* THANKS!

\*/

/\*\*

\* 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

\*/

/\*\*

\* Starting number for renaming of the files in the source folder

\* Change this to the desired starting number of the unique ID

\* Unique ID of the Student

\*

\* Then the prefix of the files after renaming

\* Right now it is NULL because we don't want anything

\*/

var STARTING\_NUMBER; // Change this to the desired starting number

var sheetName = "PeerEval";

/\*\*

\* Fetching the Starting\_Number for the code by itself rather than taking the user input

\*/

function getUniqueIDStart(){

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

STARTING\_NUMBER = sheet.getRange('C2').getValue();

Logger.log(STARTING\_NUMBER);

}

/\*\*

\* Files are renamed in the source folder

\* According to the naming conventions defined

\* Naming Conventions can be very easily changed

\*/

// Re-Naming Section

function renameFilesInFolder(source\_folder, target\_folder, students\_per\_batch, num\_Questions) {

/\*\*

\* This function is supposed to change the name of the

\* files in the @SourceFolder to the roll number

\* based on the @STARTING\_NUMBER given in Global Variable

\*

\* Code Working Algorithm

\* - The files are stored in an array in their order of name

\* - If the files are not in the order, it will sort them

\* - It will then start to rename them in that order

\*

\* Output

\* - the output file name will be of the type

\* - "501.pdf" where the 501 comes from the @STARTING\_NUMBER

\* and the ".pdf" comes from the @newFileName

\*

\* TO be used in the MAIN Driver Code

\*/

var folder = DriveApp.getFolderById(source\_folder);

var files = folder.getFiles();

// Array to store file names for sorting

var fileNames = [];

// Iterate through each file in the folder and store the names

while (files.hasNext()) {

var file = files.next();

fileNames.push(file.getName());

}

// Sort file names in alphabetical order

fileNames.sort();

// Iterate through sorted file names and rename files

for (var i = 0; i < fileNames.length; i++) {

var file = folder.getFilesByName(fileNames[i]).next();

// Increment the starting number for each file

var newNumber = STARTING\_NUMBER + i;

// Construct the new file name

var newFileName = newNumber + '.pdf';

// Rename the file

file.setName(newFileName);

}

Logger.log('Renamed ' + fileNames.length + ' files.');

}

/\*\*

\* 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.moveTo(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);

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);

}

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

}

/\*\*

\* Send Mail to the students about the folder access and the spreadsheet access

\* This is for them to evaluate their peers and give them the marks

\*/

function sendEmails(source\_folder, target\_folder, students\_per\_batch, num\_Questions) {

/\*\*

\* This code iterates through to the last column in the sheet and checks the Email-IDs

\* Then it sends the links to them individually.

\*

\* Change the message before final implementation

\*

\* To be part of the MAIN Driver Code

\*/

var workbook = SpreadsheetApp.getActiveSpreadsheet();

var sheet = workbook.getSheetByName(sheetName);

if (!sheet) {

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

return;

}

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

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

var name = data[i][0];

var email = data[i][1]; // Column B

var editorLink1 = data[i][5]; // Column F (assuming it contains the first editor link)

var lastColumn = data[0].length - 1; // Exclude the last column which may be a label

// Find the last column with data in the row

while (lastColumn > 0 && !data[i][lastColumn]) {

lastColumn--;

}

var editorLink2 = data[i][lastColumn]; // Last column with data

var subject = 'Peer Evaluation Link';

var message = 'Dear ' + name + ',\n\n' +

'Find below the link for the folder of answer scripts (view only) and the Evaluation Sheet (edit access) for the same:\n\n' +

'Folder Link: ' + editorLink1 + '\n' +

'Evaluation Sheet Link: ' + editorLink2 + '\n\n' +

'Kind Regards,\nStepWISE\n\nNOTE: This is a test mail, kindly check link activity and report anomalies.';

// Send email

MailApp.sendEmail(email, subject, message);

Logger.log(editorLink2);

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

}

Logger.log('All emails sent.');

}

/\*\*

\* The Final Driver Code MAIN

\* The Boss Code.

\* All Functionalities are to be called from this code

\*/

function mainPreEval(source\_folder, target\_folder, students\_per\_batch, 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 mainProcess...');

Logger.log(num\_Questions + '1');

// just because someone could not set the unique ID by hand

try {

Logger.log('Fetching the Unique ID starting number');

getUniqueIDStart();

Logger.log('Unique ID starter found');

} catch (e) {

Logger.log('Error in setting the Unique ID: ' + e.toString);

return;

}

// Renaming the foiles in the source folder

try {

Logger.log('Renaming files in the source folder...');

renameFilesInFolder(source\_folder, target\_folder, students\_per\_batch, num\_Questions);

Logger.log('Files renamed successfully.');

} catch (e) {

Logger.log('Error renaming files: ' + e.toString());

return;

}

//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;

}

// Sending the mail to the students with the links

try {

Logger.log("Starting to send the mails");

sendEmails(source\_folder, target\_folder, students\_per\_batch, num\_Questions);

Logger.log("Completed sending the mails.");

} catch (e) {

Logger.log("Error in sending Mail. 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);

}

Post-Evaluation Google App Script

/\*\*

\* @name PEER EVALUATION AUTOMATION (Pre-Evaluation System)

\*

\* @fileoverview The code is made to accomodate the Peer Evaluation System throughout the G-suit.

\* will include the Googel Drive and Google Sheets

\*

\*

\* @changelog

\* -version beta

\* -to be tested and cosmetic changes to be made.

\* - released 21-06-2024

\*/

/\*\*

\* BIG NOTE:

\*

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\* CHECK or CHANGE THE GLOBAL VARIABLES BEFORE EVERY RUN

\*

\* THANKS!

\*/

/\*\*

\* 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.');

}

Graph Generating Google App Script

function runAllChartFunctions() {

var spreadsheet = SpreadsheetApp.getActiveSpreadsheet();

var sheetName = 'Evaluation Results'; // Replace with your sheet name

var sheet = spreadsheet.getSheetByName(sheetName);

if (!sheet) {

Logger.log("Sheet '" + sheetName + "' not found.");

return;

}

createStandardDeviationChart(sheet, 10, 1); // Example position

createAverageScoresChart(sheet, 30, 1); // Example position

createIndividualPeerScoresChart(sheet, 10, 7); // Example position

createScoresClusteredColumnChart(sheet, 30, 7); // Example position

createScatterPlotAvgVsStdDev(sheet, 50, 4); // Example position

}

function createStandardDeviationChart(sheet, row, column) {

if (!sheet) {

Logger.log("Sheet is undefined in createStandardDeviationChart.");

return;

}

var stdRowData = sheet.getRange('B5:I5').getValues()[0]; // Adjust range as needed

var chartData = [['Student', 'Standard Deviation']];

for (var i = 0; i < stdRowData.length; i++) {

chartData.push(['Student ' + (400 + i), stdRowData[i]]);

}

var chart = sheet.newChart()

.setChartType(Charts.ChartType.COLUMN)

.addRange(sheet.getRange('A5:A5')) // Range for single cell A5 (Student ID label)

.addRange(sheet.getRange('B5:I5')) // Range for standard deviation data

.setOption('title', 'Standard Deviation of Scores for Each Student')

.setOption('hAxis', { title: 'Student', slantedText: false, slantedTextAngle: 0 }) // Set title for horizontal axis (x-axis) and disable slanted text

.setOption('vAxis', { title: 'Standard Deviation' }) // Set title for vertical axis (y-axis)

.setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top

.setPosition(row, column, 0, 0) // Adjust position as needed

.build();

sheet.insertChart(chart);

}

function createAverageScoresChart(sheet, row, column) {

var avgRowData = sheet.getRange('B6:I6').getValues()[0]; // Adjust range as needed

var chartData = [['Student', 'Average Score']];

for (var i = 0; i < avgRowData.length; i++) {

chartData.push(['Student ' + (400 + i), avgRowData[i]]);

}

var chart = sheet.newChart()

.setChartType(Charts.ChartType.COLUMN)

.addRange(sheet.getRange('A6:A6'))

.addRange(sheet.getRange('B6:I6'))

.setOption('title', 'Average Scores for Each Student')

.setOption('hAxis', { title: 'Student' })

.setOption('vAxis', { title: 'Average Score' })

.setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top

.setPosition(row, column, 0, 0) // Adjust position as needed

.build();

sheet.insertChart(chart);

}

function createIndividualPeerScoresChart(sheet, row, column) {

var students = ['400', '401', '402', '403', '404', '405', '406', '407'];

var chartData = [['Student', 'Peer 1', 'Peer 2', 'TA']];

for (var i = 0; i < students.length; i++) {

var peer1Score = sheet.getRange('B2').offset(0, i).getValue();

var peer2Score = sheet.getRange('B2').offset(1, i).getValue();

var taScore = sheet.getRange('B2').offset(2, i).getValue();

chartData.push([students[i], peer1Score, peer2Score, taScore]);

}

// Clear a temporary range to hold the chart data

var tempRange = sheet.getRange('A10:D18');

tempRange.clearContent();

tempRange.setValues(chartData);

var chart = sheet.newChart()

.setChartType(Charts.ChartType.COLUMN)

.addRange(tempRange)

.setOption('title', 'Scores Given by Each Peer and TA')

.setOption('hAxis', { title: 'Student' })

.setOption('vAxis', { title: 'Score' })

.setOption('series', {

0: { color: 'blue', labelInLegend: 'Peer 1' },

1: { color: 'red', labelInLegend: 'Peer 2' },

2: { color: 'green', labelInLegend: 'TA' }

})

.setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top

.setOption('isStacked', false)

.setPosition(row, column, 0, 0) // Adjust position as needed

.build();

sheet.insertChart(chart);

}

function createScoresClusteredColumnChart(sheet, row, column) {

// Example data range:

// Column A: Assessment/Time Point

// Columns B-H: Scores for Students 400-407

// Adjust these ranges based on your actual data layout

var dataRange = sheet.getRange('A2:I6'); // Adjust this range as needed

var chart = sheet.newChart()

.setChartType(Charts.ChartType.COLUMN)

.addRange(dataRange)

.setOption('title', 'Clustered Scores Across Students')

.setOption('hAxis', { title: 'Assessment/Time Point' })

.setOption('vAxis', { title: 'Score' })

.setOption('legend', { position: 'top', alignment: 'center' }) // Add legend to the top

.setOption('series', {

0: { color: 'blue', labelInLegend: 'Student 400' },

1: { color: 'red', labelInLegend: 'Student 401' },

2: { color: 'green', labelInLegend: 'Student 402' },

3: { color: 'orange', labelInLegend: 'Student 403' },

4: { color: 'purple', labelInLegend: 'Student 404' },

5: { color: 'cyan', labelInLegend: 'Student 405' },

6: { color: 'magenta', labelInLegend: 'Student 406' },

7: { color: 'gray', labelInLegend: 'Student 407' }

})

.setPosition(row, column, 0, 0) // Adjust position as needed

.build();

sheet.insertChart(chart);

}

function createScatterPlotAvgVsStdDev(sheet, row, column) {

var avgRange = sheet.getRange('B6:I6'); // Average scores range

var stdDevRange = sheet.getRange('B5:I5'); // Standard deviation range

var avgData = avgRange.getValues()[0];

var stdDevData = stdDevRange.getValues()[0];

var scatterData = [['Average Score', 'Standard Deviation']];

for (var i = 0; i < avgData.length; i++) {

scatterData.push([avgData[i], stdDevData[i]]);

}

// Clear a temporary range to hold the chart data

var tempRange = sheet.getRange('A30:B38');

tempRange.clearContent();

tempRange.setValues(scatterData);

var chart = sheet.newChart()

.setChartType(Charts.ChartType.SCATTER)

.addRange(tempRange)

.setOption('title', 'Average Score vs. Standard Deviation')

.setOption('hAxis', { title: 'Average Score' })

.setOption('vAxis', { title: 'Standard Deviation' })

.setOption('legend', { position: 'none' }) // Hide legend for scatter plot

.setPosition(row, column, 0, 0) // Adjust position as needed

.build();

sheet.insertChart(chart);

}

Button Generating Google App Script

function onOpen() {

/\*\*

\* Making the buttons

\*

\* When copied and pasted for replication of the sheet

\* Just run this function to make everything.

\*/

var ui = SpreadsheetApp.getUi();

ui.createMenu('Peer Evaluation')

.addItem('Set Parameters', 'showInputDialog')

.addItem('Pre Evaluation', 'runMainPreEval')

.addItem('Post Evaluation', 'runMainPostEval')

// .addItem('Generate Charts', 'generateCharts')

.addToUi();

ui.createMenu('Graphs')

.addItem('Generate Charts', 'generateCharts')

.addToUi();

}

function showInputDialog() {

/\*\*

\* Set up of the Peer Evaluation Menu Button Sub-buttons

\*

\* Uses Properties Service of the Document to save Global vairables

\* which will be accessed later by the runMainPreEval and runMainPostEval

\*/

var ui = SpreadsheetApp.getUi();

var response1 = ui.prompt('Source Folder ID', 'Please enter the Source Folder ID', ui.ButtonSet.OK\_CANCEL);

if (response1.getSelectedButton() == ui.Button.OK) {

PropertiesService.getDocumentProperties().setProperty('SOURCE\_FOLDER\_ID', response1.getResponseText());

} else {

ui.alert('Operation cancelled.');

return;

}

var response2 = ui.prompt('Target Folder ID', 'Please enter the Target Folder ID', ui.ButtonSet.OK\_CANCEL);

if (response2.getSelectedButton() == ui.Button.OK) {

PropertiesService.getDocumentProperties().setProperty('TARGET\_FOLDER\_ID', response2.getResponseText());

} else {

ui.alert('Operation cancelled.');

return;

}

var response3 = ui.prompt('Number of Students Per Batch', 'How many students in eacah batch?', ui.ButtonSet.OK\_CANCEL);

if (response3.getSelectedButton() == ui.Button.OK) {

PropertiesService.getDocumentProperties().setProperty('STUDENTS\_PER\_FOLDER', response3.getResponseText());

} else {

ui.alert('Operation cancelled.');

return;

}

var response4 = ui.prompt('Number of Questions', 'How many questions in the paper?', ui.ButtonSet.OK\_CANCEL);

if (response4.getSelectedButton() == ui.Button.OK) {

PropertiesService.getDocumentProperties().setProperty('numQuestions', response4.getResponseText());

} else {

ui.alert('Operation cancelled.');

return;

}

var properties = PropertiesService.getDocumentProperties();

var source\_folder = properties.getProperty('SOURCE\_FOLDER\_ID');

var target\_folder = properties.getProperty('TARGET\_FOLDER\_ID');

var students\_per\_batch = properties.getProperty('STUDENTS\_PER\_FOLDER');

var num\_Questions = properties.getProperty('numQuestions');

ui.alert('Parameters set: ' + '\nSource Folder: ' + source\_folder + '\nTarget Folder: ' + target\_folder

+ '\nStudents per Batch: ' + students\_per\_batch + '\nNumber of Questions: ' + num\_Questions);

}

function runMainPreEval() {

/\*\*

\* Accesses the Document Property Services and calls the mainPreEval

\* with the global variables as the function parameters

\*/

var properties = PropertiesService.getDocumentProperties();

var source\_folder = properties.getProperty('SOURCE\_FOLDER\_ID');

var target\_folder = properties.getProperty('TARGET\_FOLDER\_ID');

var students\_per\_batch = properties.getProperty('STUDENTS\_PER\_FOLDER');

var num\_Questions = properties.getProperty('numQuestions');

if (source\_folder === '' || target\_folder === '' || students\_per\_batch === '' || num\_Questions === '') {

SpreadsheetApp.getUi().alert('Please set the parameters first using the "Set Parameters" menu item.');

return;

}

// Calling the mainPreEval to run all the necessary functions

mainPreEval(source\_folder, target\_folder, students\_per\_batch, num\_Questions);

}

function runMainPostEval() {

/\*\*

\* Accesses the Document Property Services and calls the mainPostEval

\* with the global variables as the function parameters

\*/

var properties = PropertiesService.getDocumentProperties();

var source\_folder = properties.getProperty('SOURCE\_FOLDER\_ID');

var target\_folder = properties.getProperty('TARGET\_FOLDER\_ID');

var students\_per\_batch = properties.getProperty('STUDENTS\_PER\_FOLDER');

var num\_Questions = properties.getProperty('numQuestions');

if (source\_folder === '' || target\_folder === '' || students\_per\_batch === '' || num\_Questions === '') {

SpreadsheetApp.getUi().alert('Please set the parameters first using the "Set Parameters" menu item.');

return;

}

// Calling the mainPreEval to run all the necessary functions

mainPostEval(num\_Questions);

}

function generateCharts() {

// Call the function from Graph.gs to generate charts

runAllChartFunctions();

}