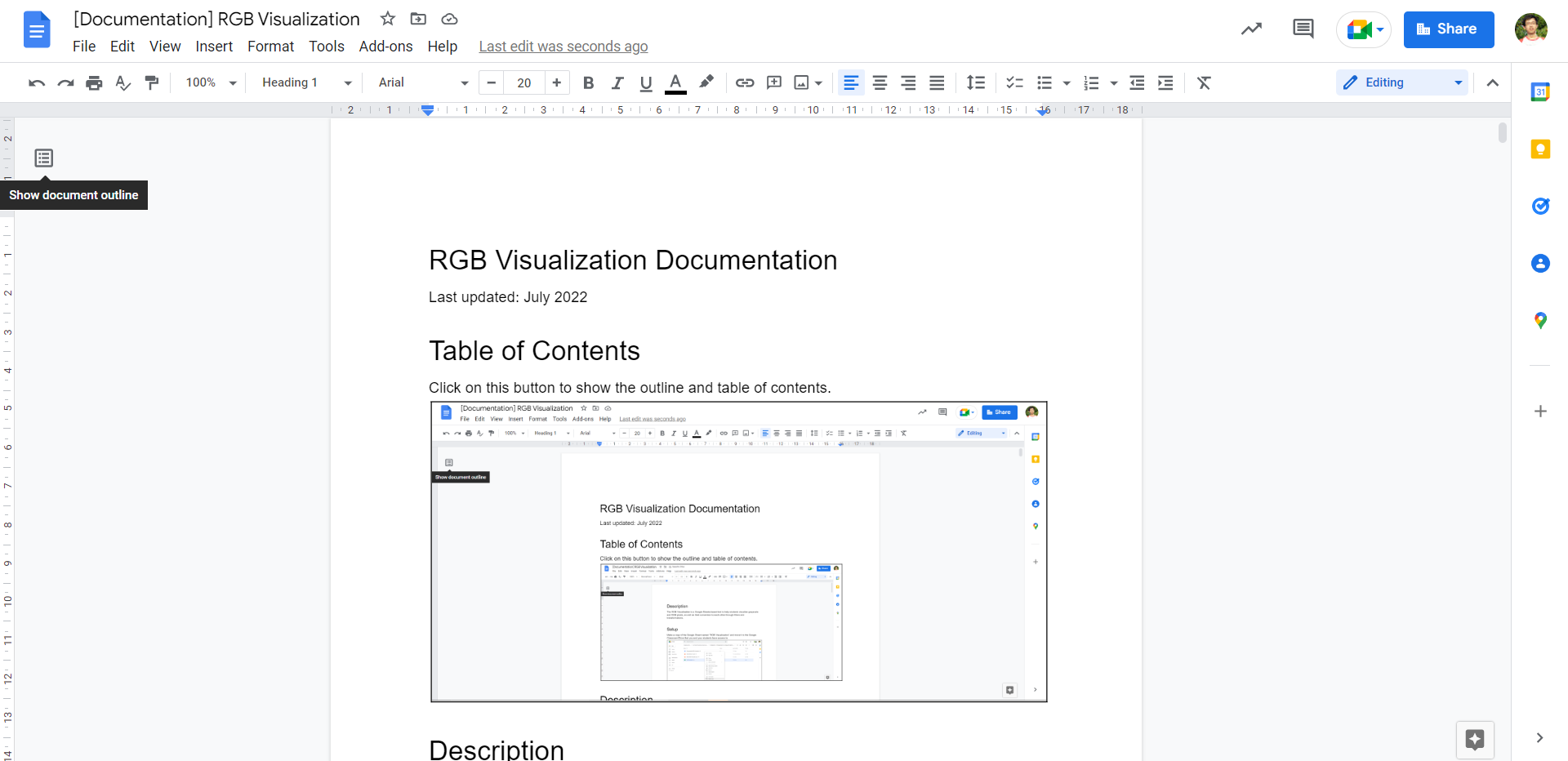
# RGB Visualization Documentation

Last updated: July 2022

# Table of Contents

Click on this button to show the outline and table of contents.

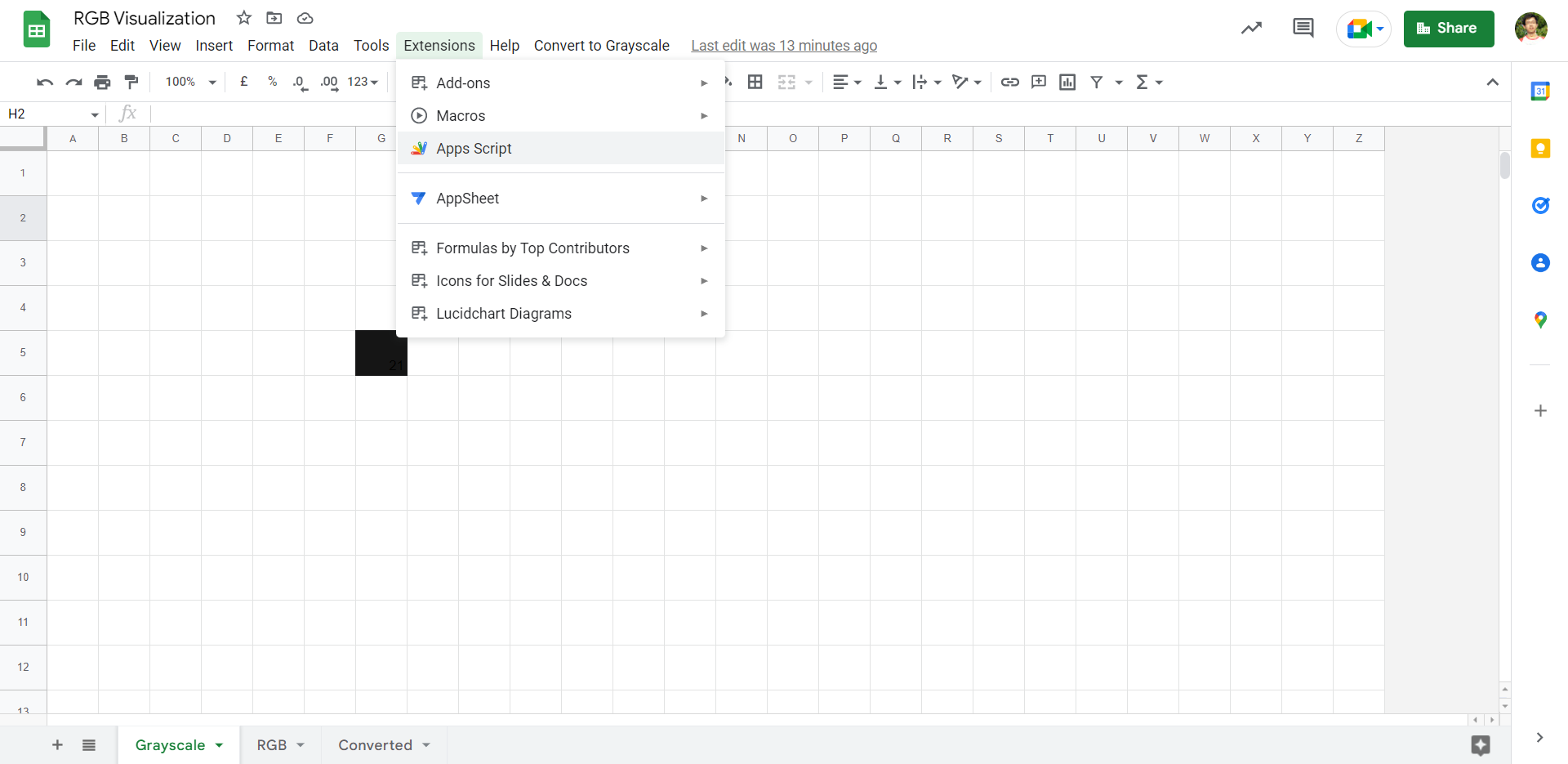
# Description

The RGB Visualization is a Google Sheets-based tool to help students visualize grayscale and RGB pixels, as well as their conversion to each other through filters and transformations.

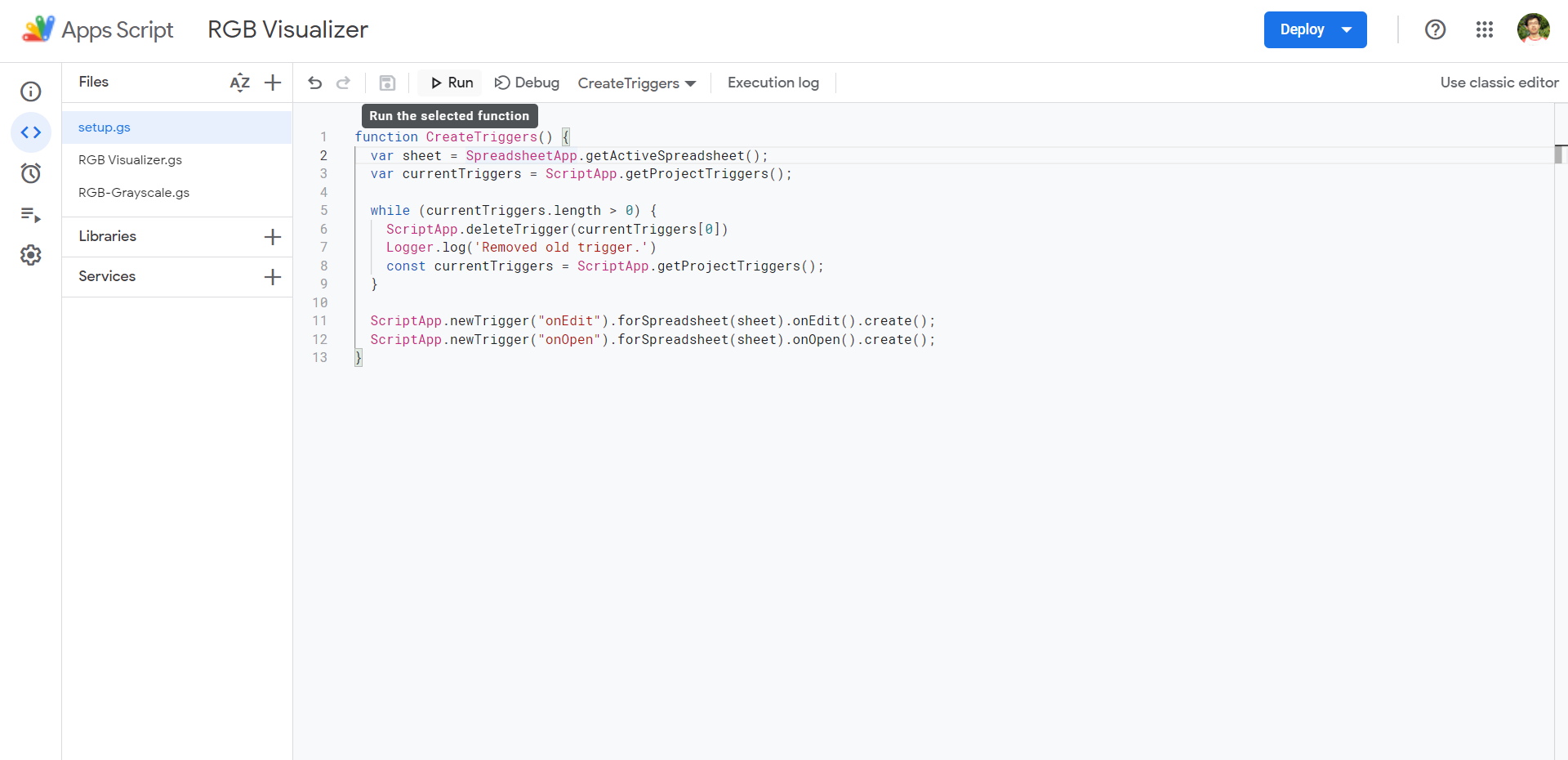
# Setup

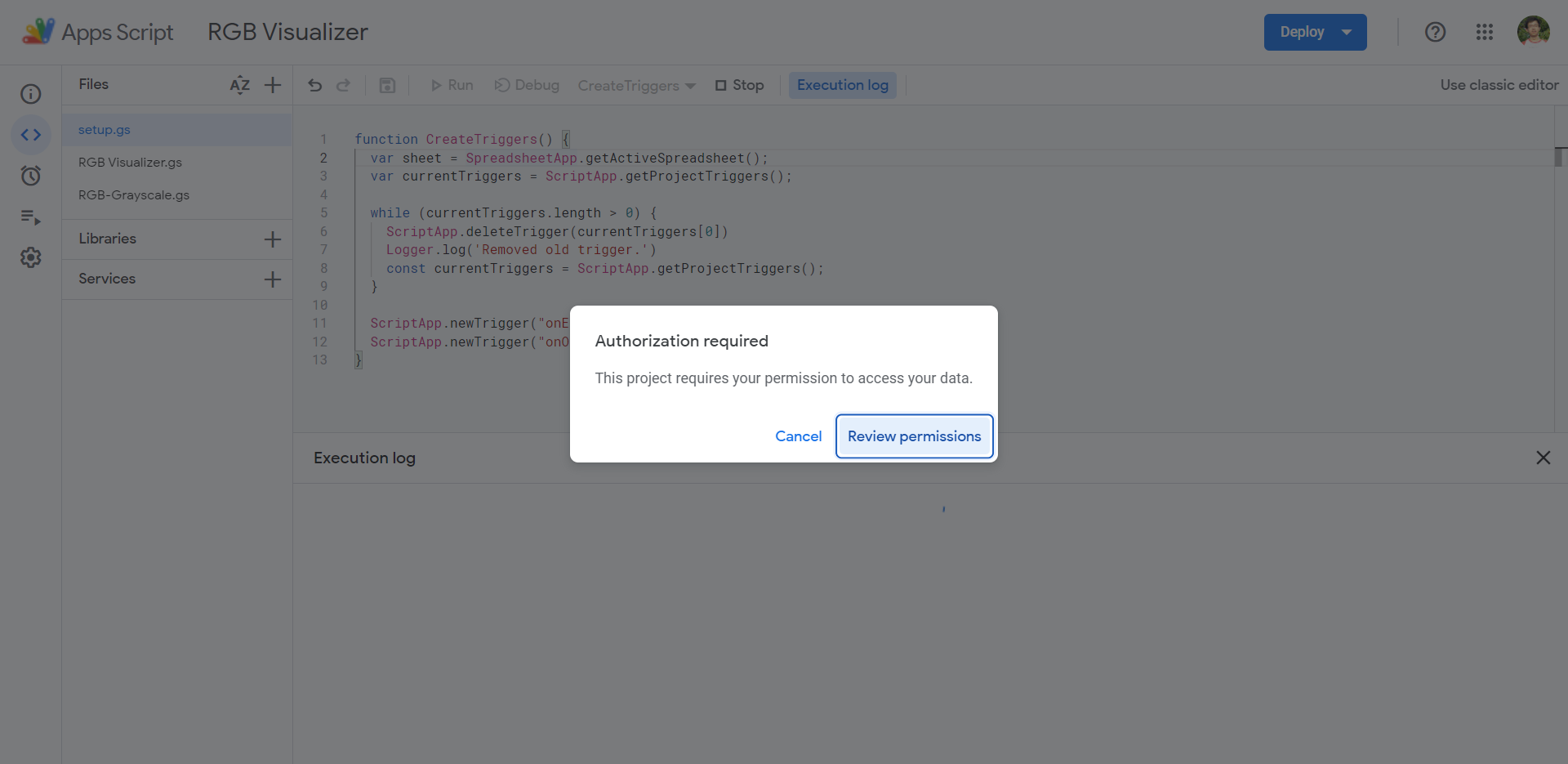
Make a copy of the Google Sheet named “RGB Visualization” and move it to the Google Classroom/Drive that you and your students have access to.

[*Image Removed due to Contractual Obligations*]

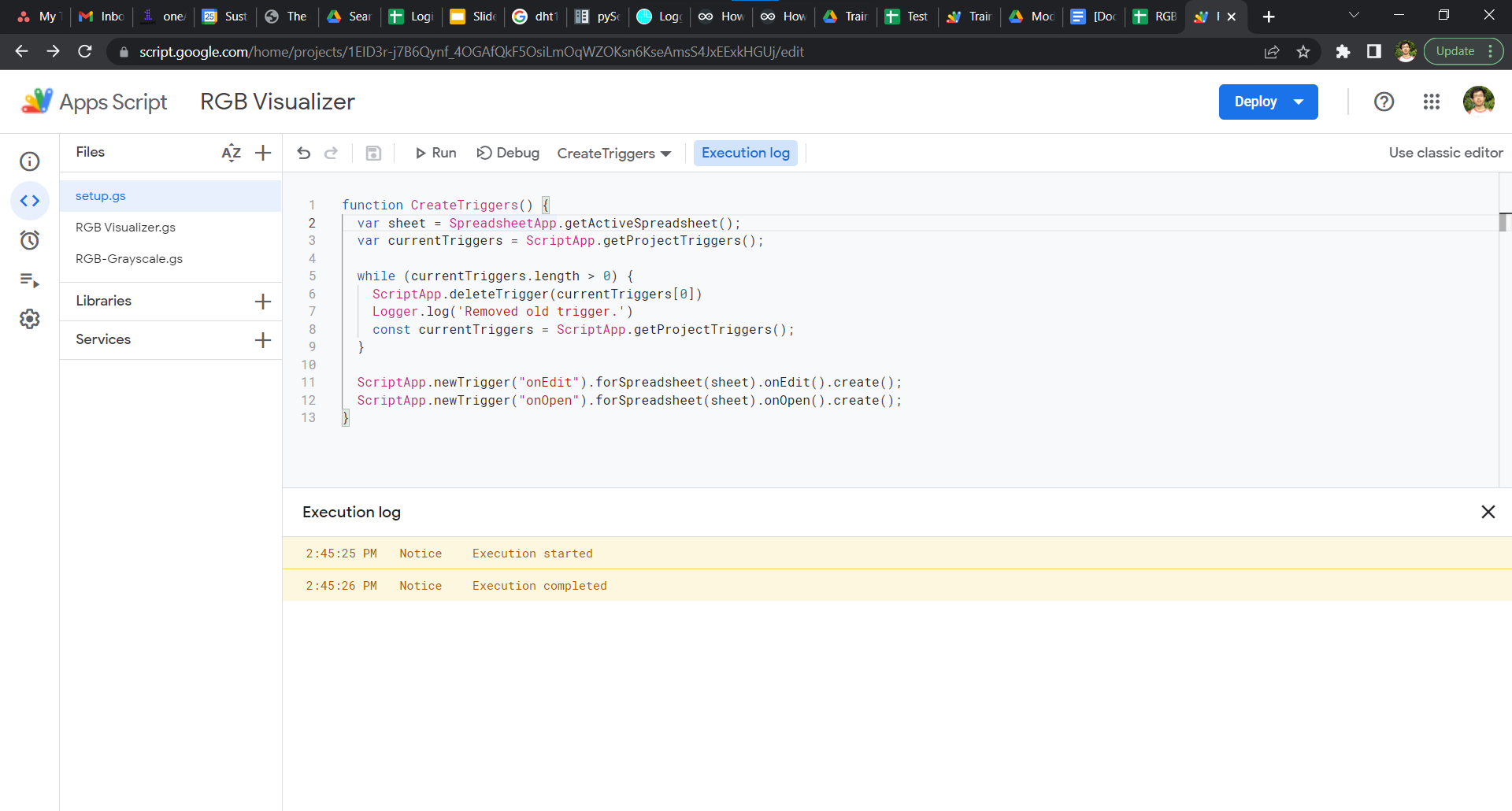
Open the Google Sheet and go to Extensions > Apps Script to open up the Apps Script Editor.

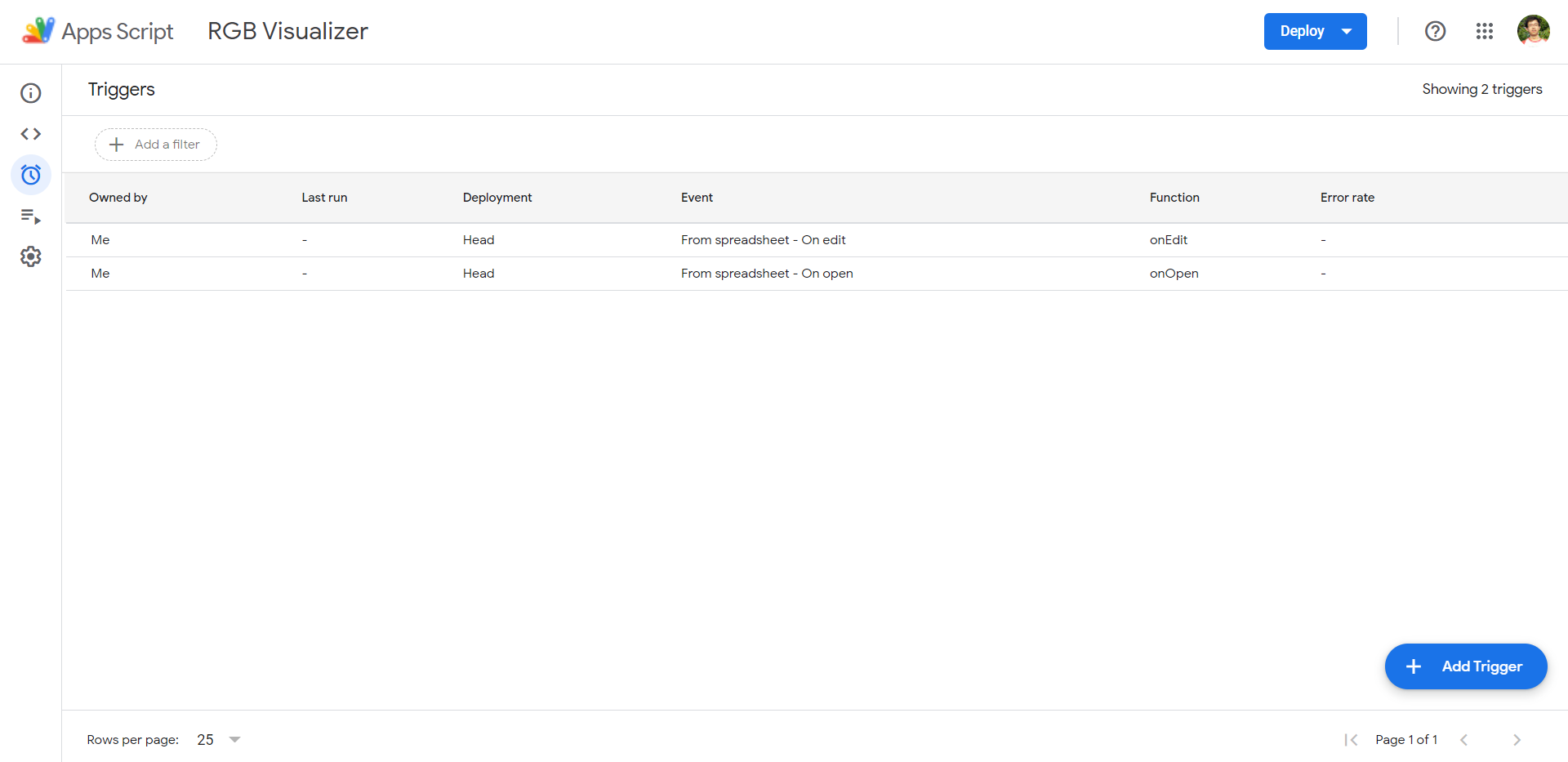
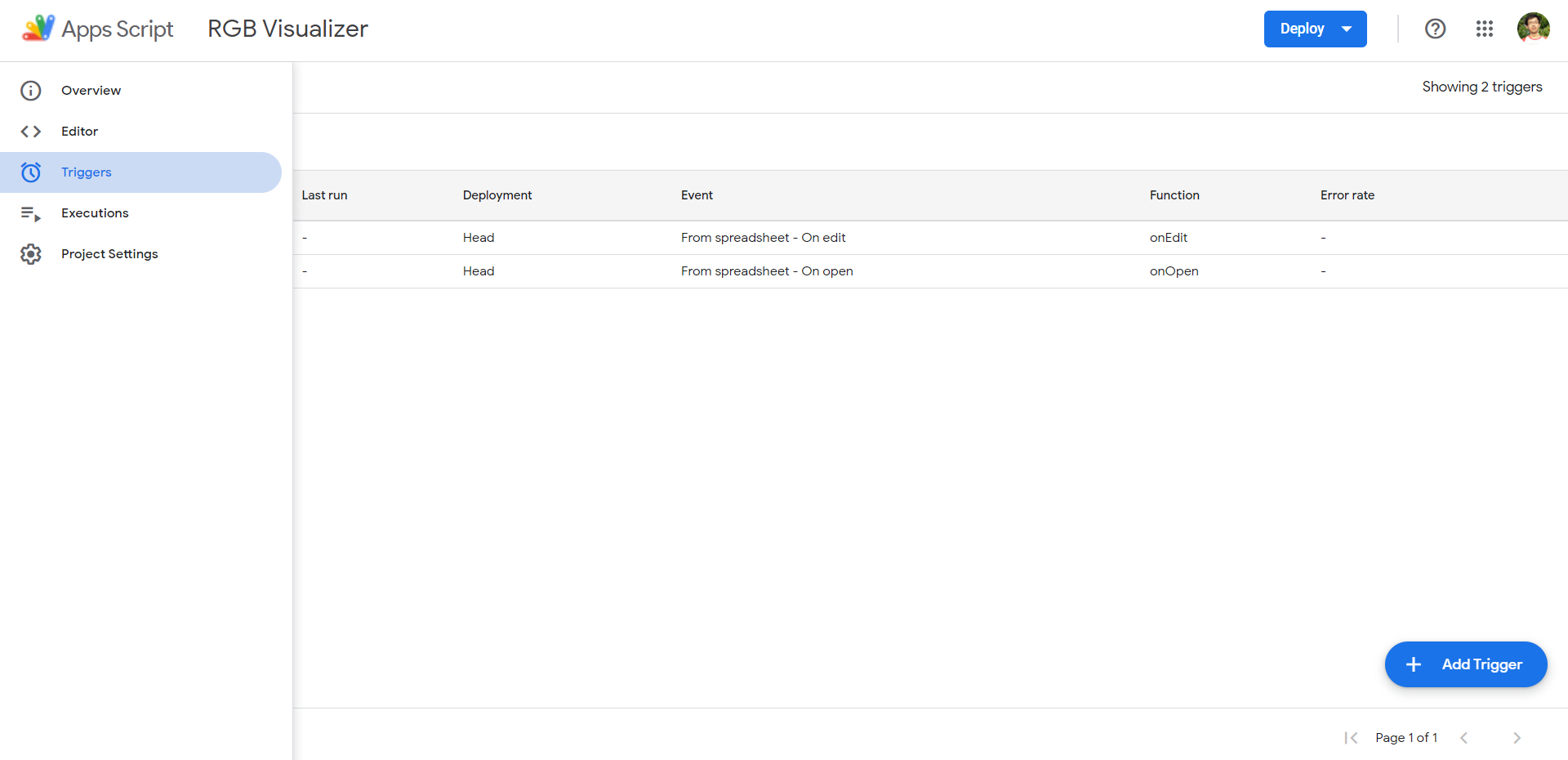
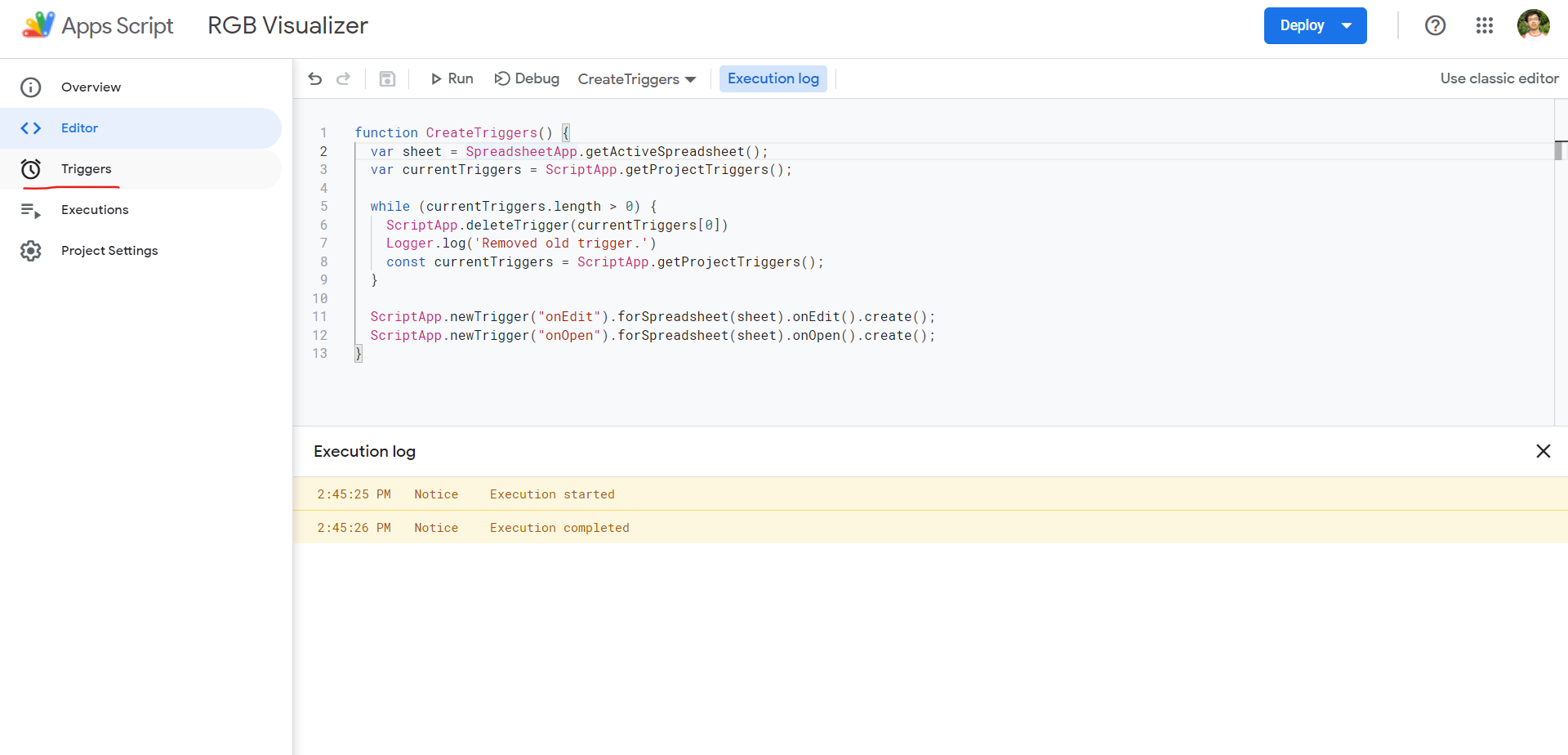
Go to the Apps Script Editor and run the CreateTriggers function within setup.gs (should be selected by default). Follow the instructions to grant the Google Sheet the necessary authorizations.

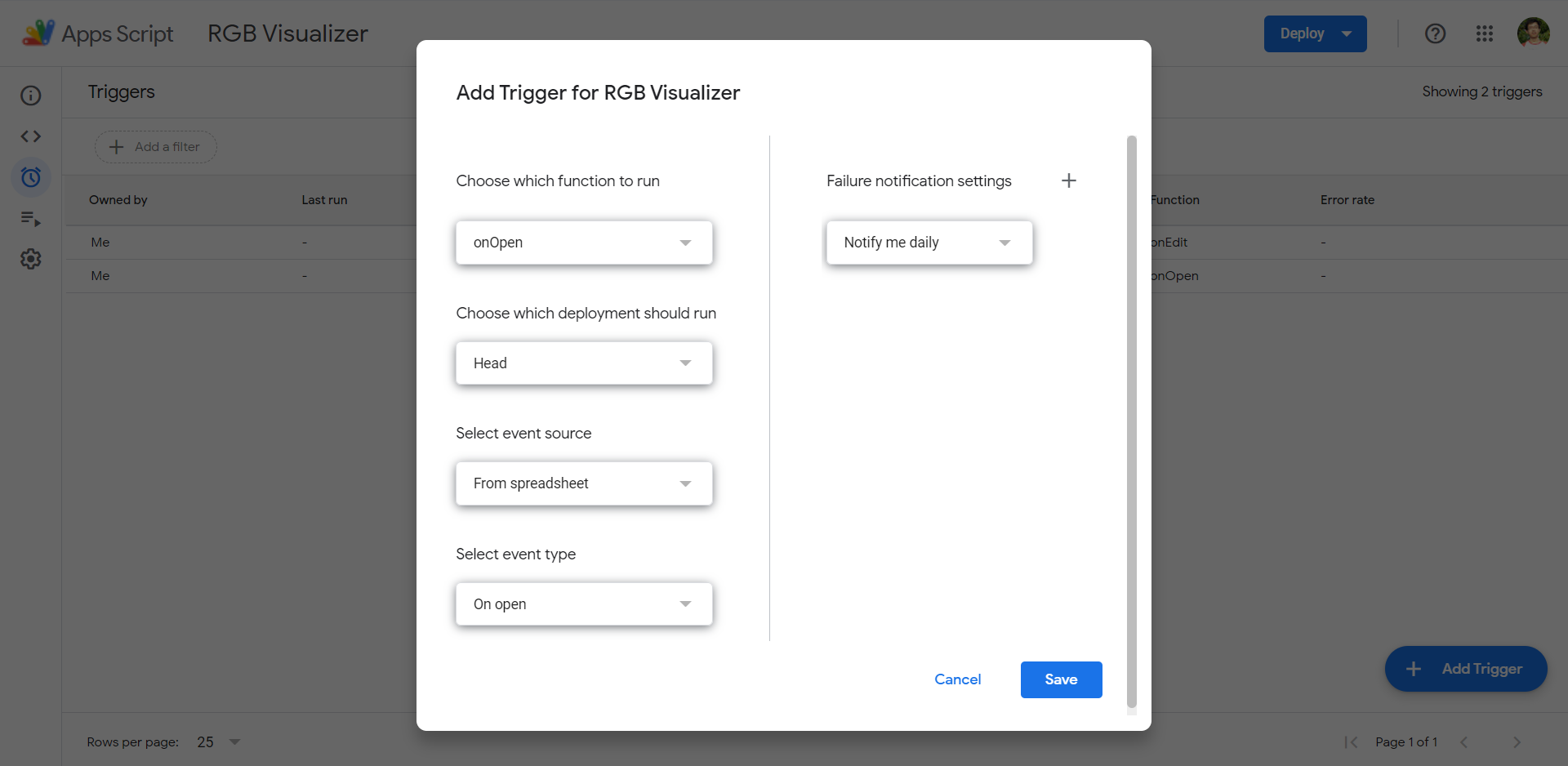
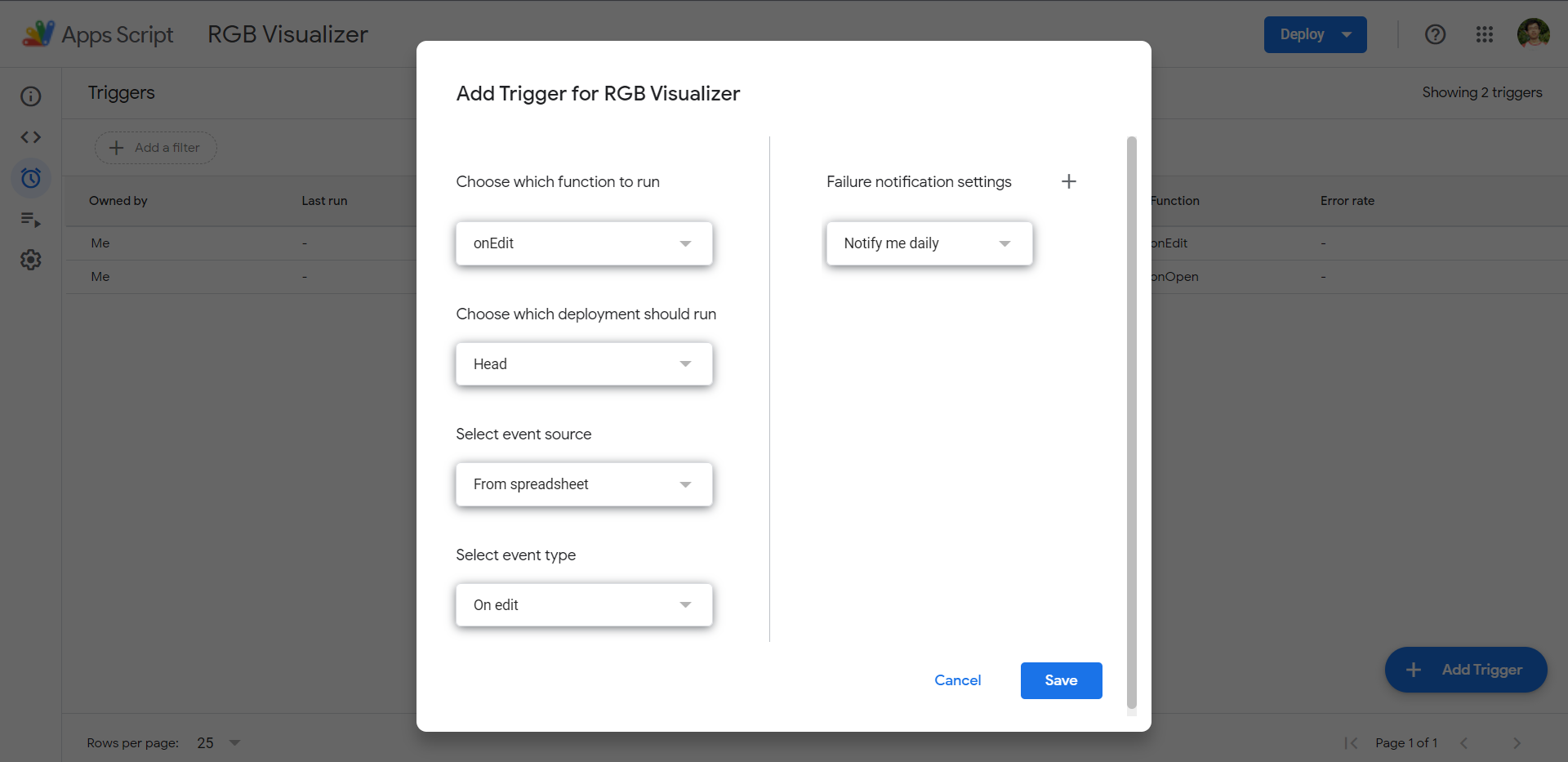




[*Images removed due to Contractual Obligations*]

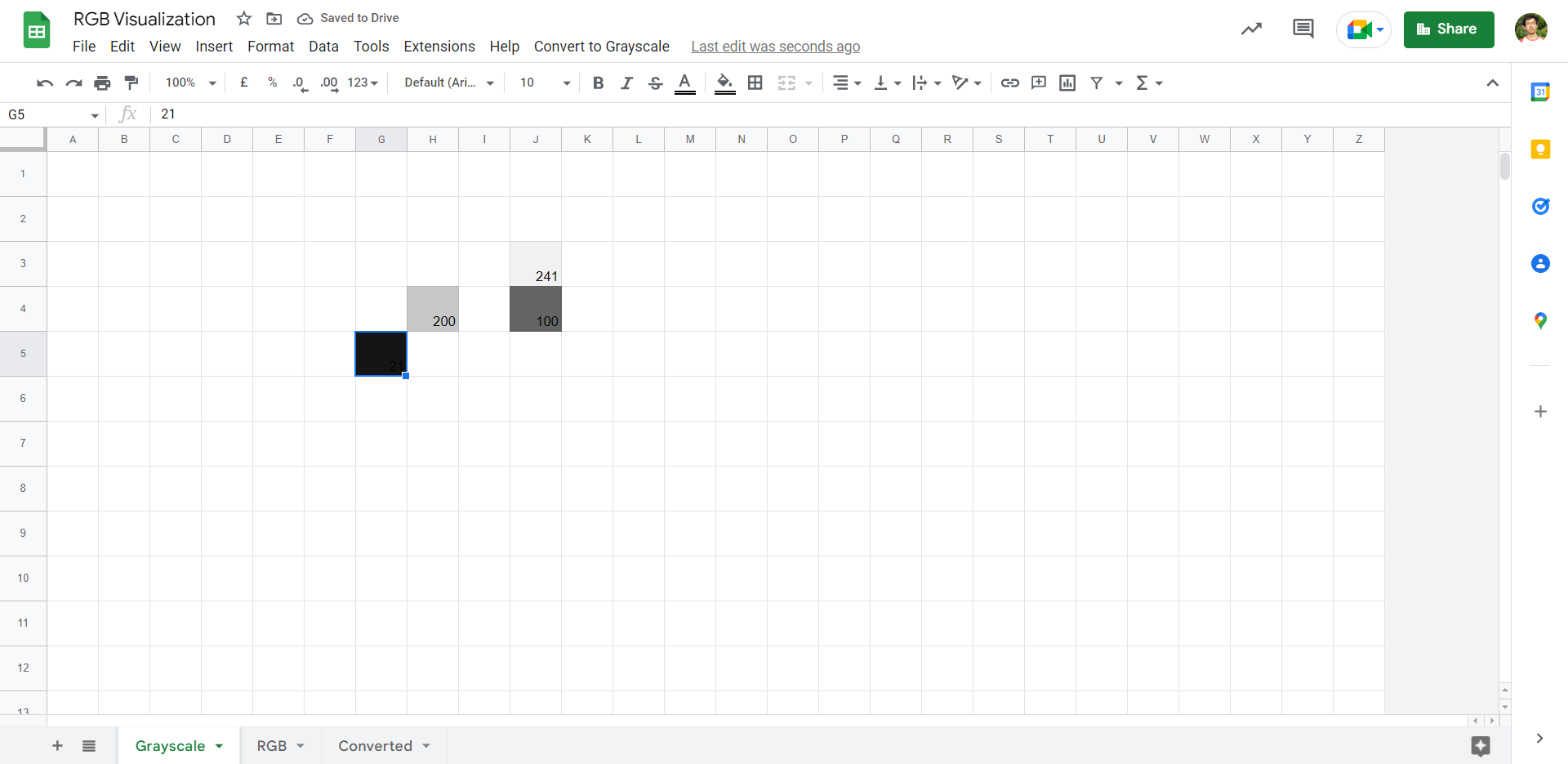


After the execution of the script is completed, you can verify that the setup was successful by checking the Triggers tab and ensuring that it is as follows.

You have completed the setup. In the event that the Triggers do not appear to be as in the figures above, repeat the CreateTriggers function or manually create the triggers using the blue “+ Add Trigger” button at the bottom right based on the details in the figures.

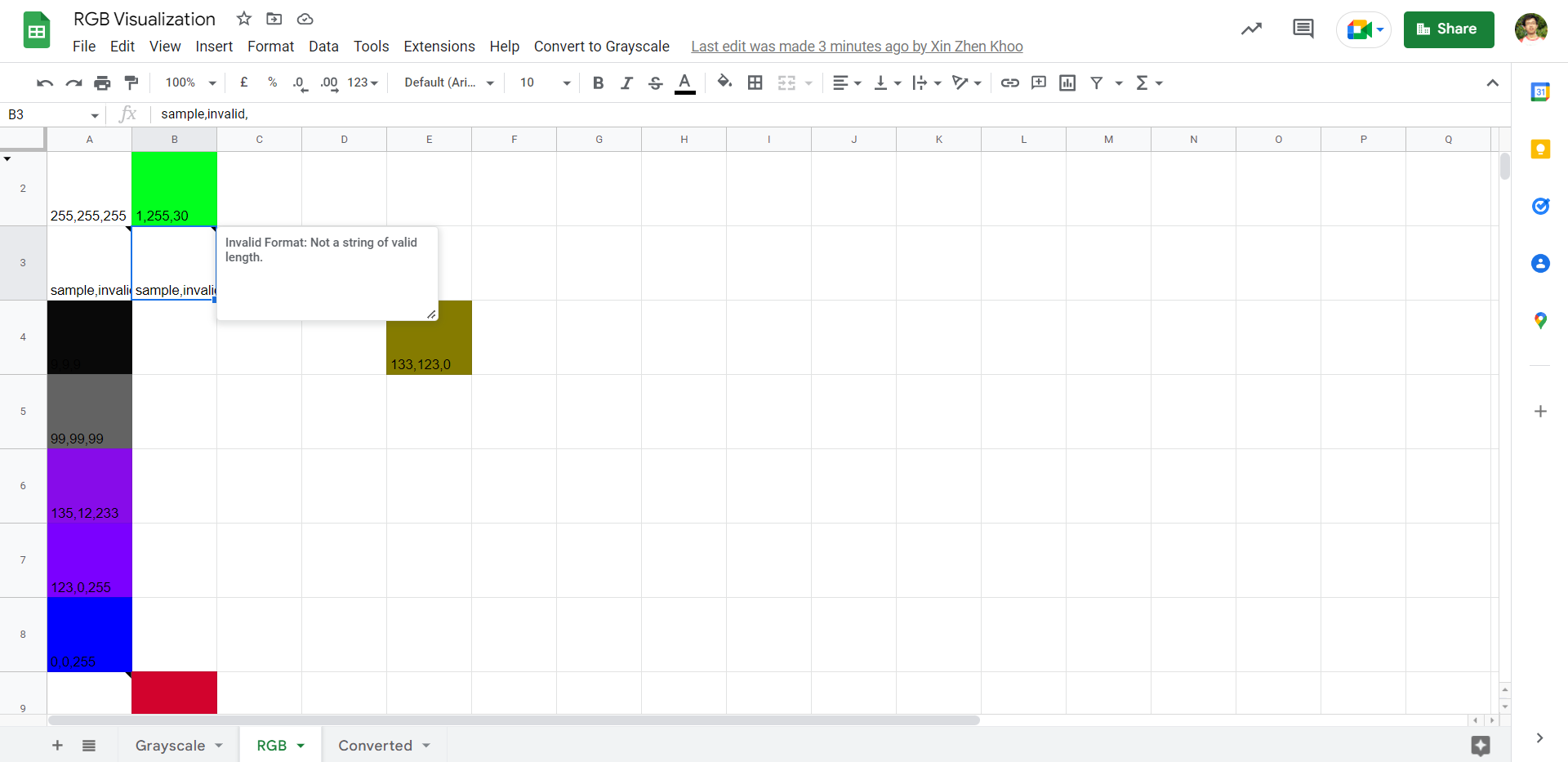
# User Guide

## Grayscale Sheet

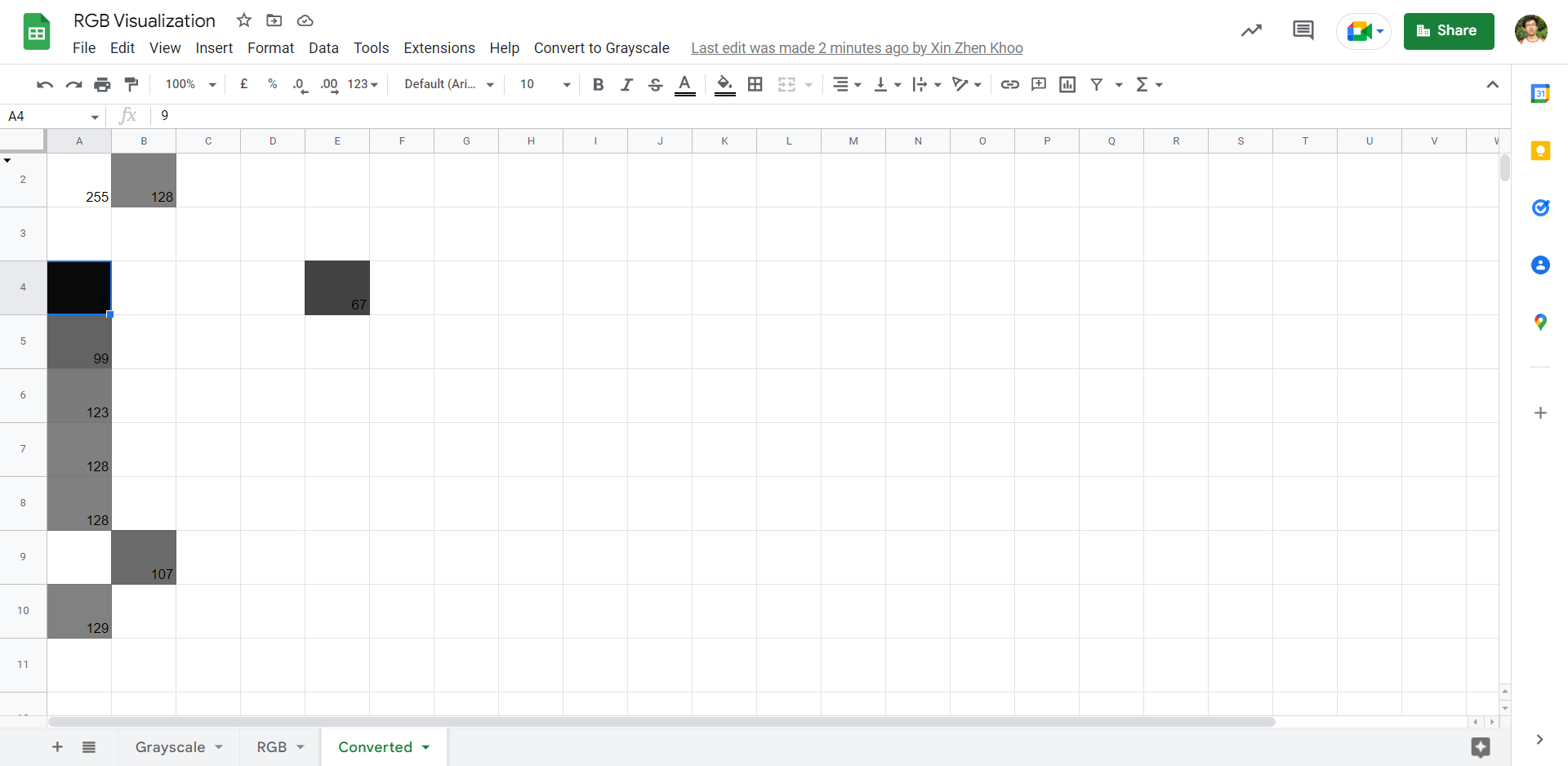
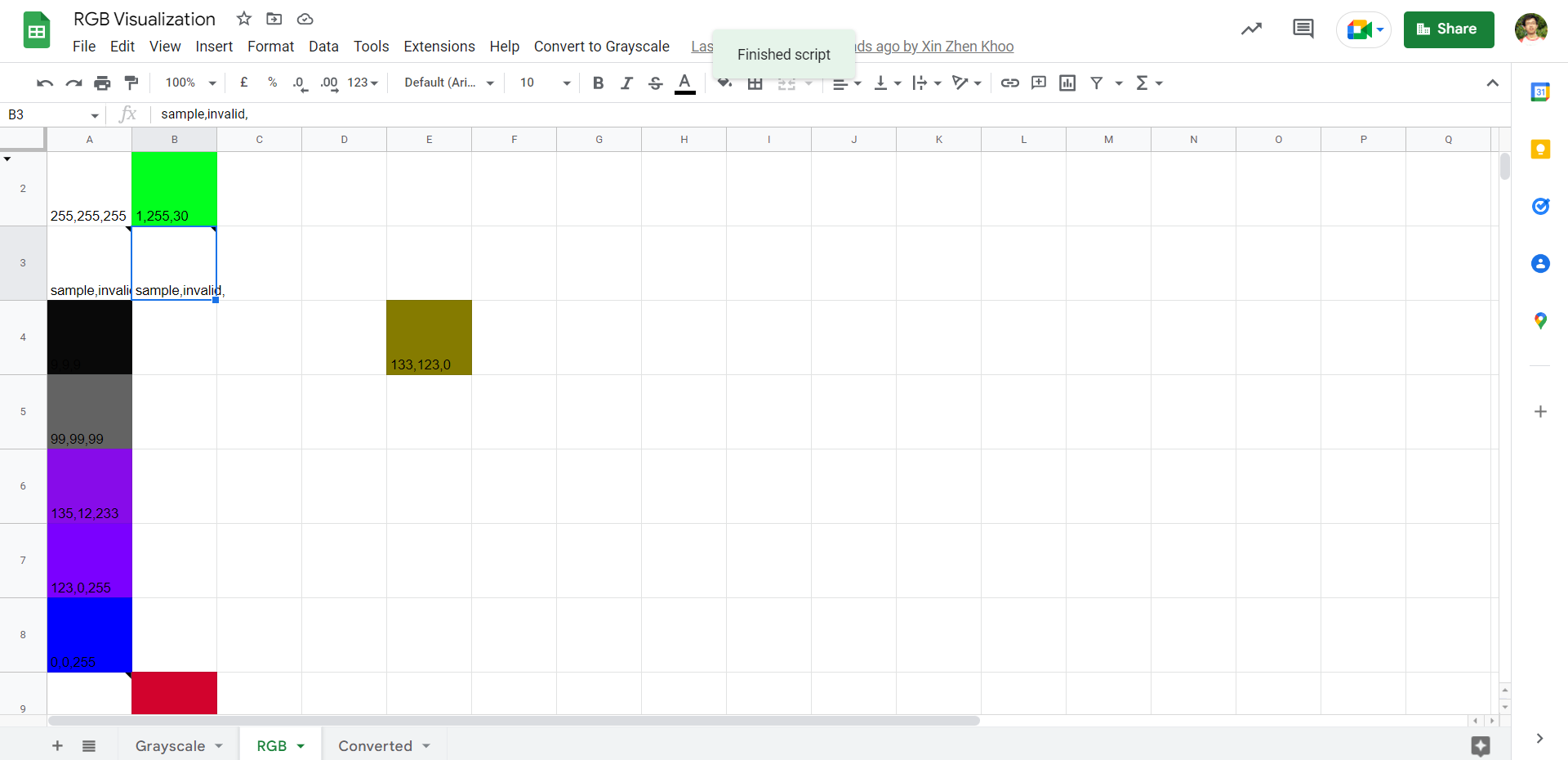
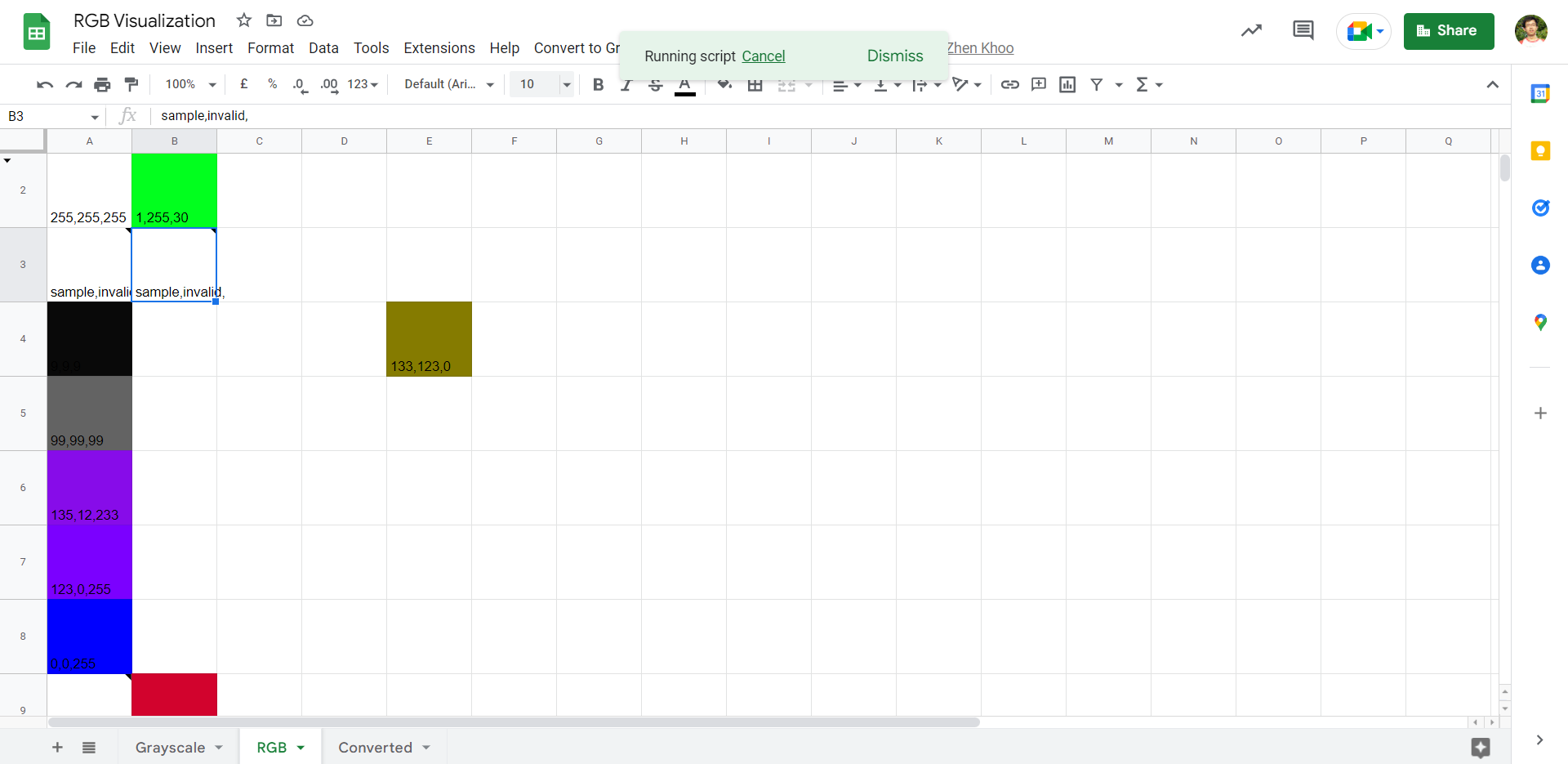
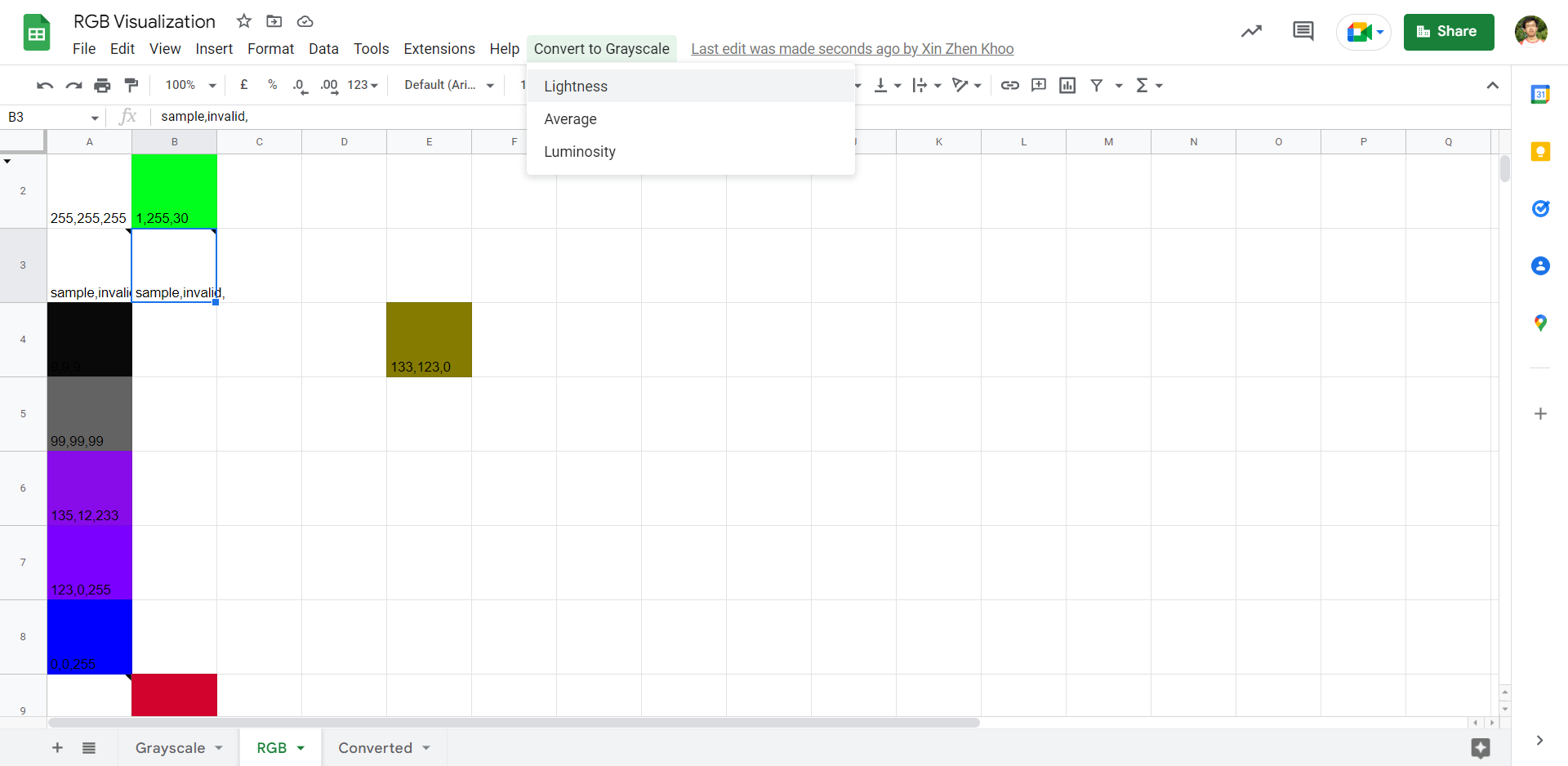
Enter numbers within the range of 0 - 255 to color the cell with the corresponding 8-bit grayscale color. Numbers outside of the range will be colored as if they were 0 or 255, depending on which side of the boundary they are outside.

## RGB Sheet

Fill in cells with the format XXX,XXX,XXX (3 sets of 1 - 3 integers separated by 2 commas) to color the cells with the corresponding 8-bit RGB. Cells filled with values of an invalid format will have a comment with the error message “Invalid Format: Not a string of valid length.”.



## Converted Sheet

The Converted Sheet displays the values and colors of the RGB Sheet after being converted using either 1 of 3 conversion schemas. To execute the conversion, click on the “Convert to Grayscale” button on the menu ribbon and select the conversion schema to convert. Invalid cell values will be ignored.

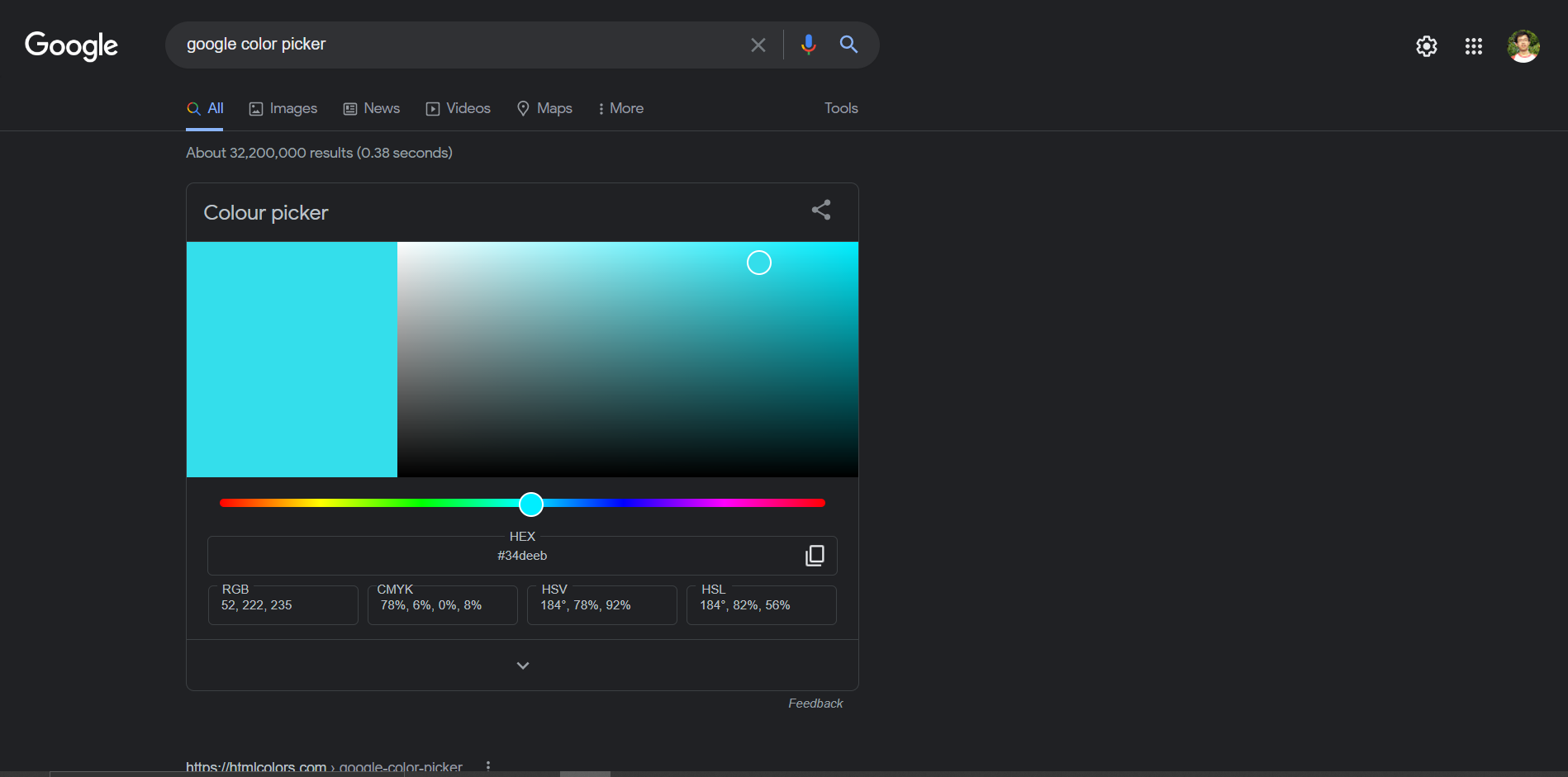
# Implementation Suggestions

In conjunction with the slides, you may use the Grayscale Sheet as a warm up exercise to increase students’ proficiency and familiarity with the tool in preparation for the similar, but slightly more complicated, RGB Sheet.

If the classroom is large, individual/small group tasks may be more suitable. Prepare multiple copies of the RGB Visualization tool or instruct students to make their own copies a la the Setup portion of this document. You may assign students to write text or recreate simple logos (like their names) with the Grayscale Sheet, followed by more complicated pixel art with the RGB Sheet.

If the classroom is relatively small, you may opt to include the entire class in one teambuilding session. Let the students get creative and wild with the Grayscale Sheet and take the opportunity to nurture their soft skills. Next, challenge the students to recreate a fairly simple colored logo and push them to increase the quality if they manage to complete it ahead of time. Using tools like Google Color Picker or similar web tools on the sample image can help increase their efficiency.

[*Image Removed due to Contractual Oblgiations*]



If there is time, you may prompt students to come up with good ways of converting RGB (3 channels) to Grayscale (1 channel). Select some of the cells/pixels and have students manually calculate the new values. Afterwards, if the students come up with similar methods as the 3 schemas provided, compare the results of their conversion and determine which method works best. You may use the “Luminosity” schema as an “optimal” option to compare against.

If there is a bit of time, you may simply demonstrate 3 different methods of converting RGB to Grayscale and compare the results using the Grayscale Sheet, from Lightness to Average to Luminosity. Lightness is the simplest method since it uses the 2 most relevant channels, Average uses all 3 channels but ignores the fact that the human eye perceives the brightness of different colors differently (hence luminosity), while Luminosity uses some common values that have been found to work well through trial-and-error by experts.

If there is no time, you may opt to skip using the Converted Sheet or simply use the Luminosity Schema to demonstrate how filters work and tie it in with the idea of applying a formulaic change to each and every pixel to achieve an overall effect (as a foundation to CNNs which will be covered in later lessons).

# Technical Documentation

Avoid sharing technical details about the tools with students before they have completed the activity to prevent disruption from curious and mischievous students. If you would like to share some of the backend details after the activity, strongly advise against touching the Apps Scripts code and make a copy of the sheet and move it to a location where they do not have access beforehand or take screenshots to save your classes’ progress/completed work.

## Setup

This tool uses several functionalities that exceed the capabilities of simple event triggers granted by Google. Hence, custom event triggers and relevant authorizations have to be granted before using it. The AppScript code saved as setup.gs serves to automate this creation process and is as follows (as a backup).

function CreateTriggers() {

var sheet = SpreadsheetApp.getActiveSpreadsheet();

var currentTriggers = ScriptApp.getProjectTriggers();

while (currentTriggers.length > 0) {

ScriptApp.deleteTrigger(currentTriggers[0])

Logger.log('Removed old trigger.')

const currentTriggers = ScriptApp.getProjectTriggers();

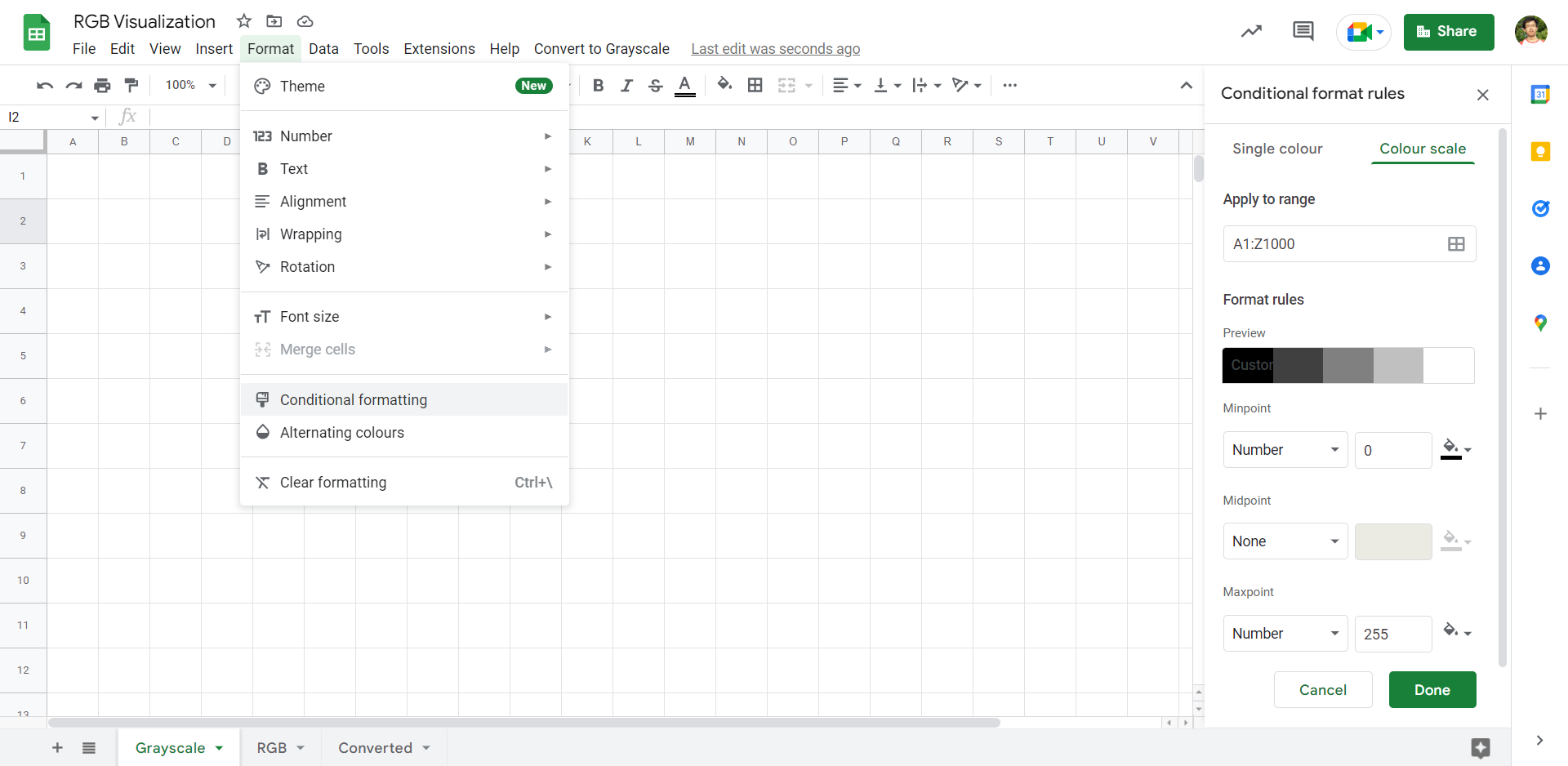
}

ScriptApp.newTrigger("onEdit").forSpreadsheet(sheet).onEdit().create();

ScriptApp.newTrigger("onOpen").forSpreadsheet(sheet).onOpen().create();

}

## Grayscale Sheet

This sheet was simply created using the conditional formatting function built into Google Sheets.

## RGB Sheet

The Apps Script code for this sheet is saved as RGB Visualizer.gs and is as follows (as a backup).

function onEdit(e) {

try {

var rng = e.range;

Logger.log(e.triggerUid);

if(rng.getSheet().getName() != 'RGB'){

Logger.log('Not RGB sheet');

Logger.log(e.range.getSheet().getName());

return 'Not RGB sheet';

}

if(rng.isBlank()){

Logger.log('Cell deleted');

rng.setBackground(null);

rng.setNote(null);

return null;

}

var clr = e.value;

var countComma = (clr.match(/,/g) || []).length;

if(clr.length <= 11 && clr.length >= 5 && countComma == 2){

let firstComma = clr.indexOf(",");

let secondComma = clr.indexOf(",",firstComma + 1)

rng.setNote(null);

rng.setBackgroundRGB(clr.slice(0,firstComma),clr.slice(firstComma + 1, secondComma), clr.slice(secondComma + 1));

}

else{

rng.setNote('Invalid Format: Not a string of valid length.');

}

}

catch(err){

Logger.log('Failed with error %s', err.message);

}

}

The script uses an onEdit event trigger to automatically detect and color valid cells within the sheet. There are some code blocks that aren’t critical, but helpful for debugging or early code exiting.

## Converted Sheet

The Apps Script code for this sheet is saved as RGB-Grayscale.gs and is as follows (as a backup).

var ui = SpreadsheetApp.getUi();

var ss = SpreadsheetApp.getActiveSpreadsheet();

var rgbSheet = ss.getSheetByName('RGB');

var convSheet = ss.getSheetByName('Converted');

function onOpen() {

ui.createMenu('Convert to Grayscale')

.addItem('Lightness', 'lgtRGB')

.addItem('Average', 'avgRGB')

.addItem('Luminosity', 'lumRGB')

.addToUi();

};

function convertRGB(type){

convSheet.getDataRange().clearContent();

var rangeData = rgbSheet.getDataRange();

var rangeNotate = rangeData.getA1Notation();

var pasteRange = convSheet.getRange(rangeNotate);

var rangeValues = rangeData.getValues();

rangeValues.forEach(rowFunc);

function rowFunc(value) {

value.forEach(cellFunc);

};

function cellFunc(clr, index, array) {

var countComma = (clr.match(/,/g) || []).length;

if(clr.length <= 11 && clr.length >= 5 && countComma == 2){

let firstComma = clr.indexOf(",");

let secondComma = clr.indexOf(",",firstComma + 1);

let r = clr.slice(0,firstComma);

let g = clr.slice(firstComma + 1, secondComma);

let b = clr.slice(secondComma + 1);

switch (type){

case 'lgt':

array[index] = Math.round((Math.min(r,g,b) + Math.max(r,g,b)) / 2);

break;

case 'avg':

array[index] = Math.round((1 \* r + 1 \* g + 1 \* b) / 3);

break;

case 'lum':

array[index] = Math.round(0.3 \* r + 0.59 \* g + 0.11 \* b);

break;

};

}else {

array[index] = '';

};

};

pasteRange.setValues(rangeValues);

};

function lgtRGB() {

convertRGB('lgt');

};

function avgRGB() {

convertRGB('avg');

};

function lumRGB() {

convertRGB('lum');

};

The script uses an onOpen event trigger to automatically create a custom menu when the Google Sheet is opened. The 3 buttons on the menu each run the corresponding function to execute the conversion using the selected schema. The 3 conversion schemas are defined within the switch-case block.

## GitHub

For archives, backups, future updates or a copy of this documentation, click on the link below to visit the creator’s GitHub repository.

[RGB Visualization @ GitHub by DiiSollertia](https://github.com/DiiSollertia/very-basic-macros/tree/main/RGB%20Visualization)