

Lab Exercise #3 - K-Means Clustering

Instruction:

Read the instructions carefully and follow each step.

Provide your best answers for all questions.

The name, email, and photo associated with your Google account will be recorded when you upload files and submit this form

* Indicates required question

Email *

Record my email address with my response

Name *

Example: Dela Cruz, Juan C.

Your answer

Student Number *

Example: 24B1234

Your answer

Section *

- 4A
- 4B
- 4C
- 4D
- 4E
- 4F

Image Compression using K-means Clustering

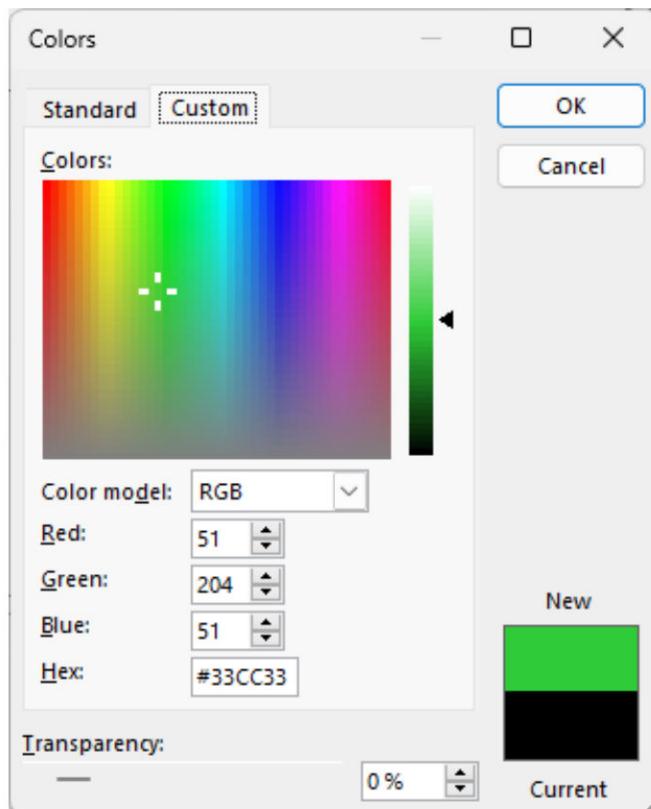
In this lab exercise, we will explore how K-means clustering can be used for image compression, a technique to significantly reduce the file size of an image without losing too much visual quality.

Background

An image is made up of pixels, the smallest units that form the picture, similar to * puzzle pieces.

In a colored image, each pixel has three values representing the intensity of Red, Green, and Blue (RGB), ranging from 0 to 255.

For example, when selecting a custom color in Microsoft Office, you can see the specific RGB values that define that color.



I have read and understood.

A 1280×720 image contains 1280×720 pixels, with each pixel requiring 24 bits, * resulting in about 22 million bits to store the image (e.g., $24 \times 1280 \times 720$).

I have read and understood.

Compression Concept

To reduce the file size, we aim to reduce the number of bits needed to represent * the image.

Using K-means clustering, we can limit the number of colors in the image.

For instance, instead of using the full range of RGB values, we could cluster the pixels into 30 dominant colors.

This reduces the storage requirement, as we need only 15 bits per pixel (5 bits per color channel) instead of 24.

I have read and understood.

By applying K-means clustering, we can reduce the file size of a 1280 x 720 * image to about 52% of its original size while preserving most of its visual quality.

I have read and understood.

Objective

In this lab, you will apply K-means clustering to compress an image by reducing * the number of colors used.

You will see how machine learning can efficiently compress image files.

I have read and understood.

Step 1: Select an Image

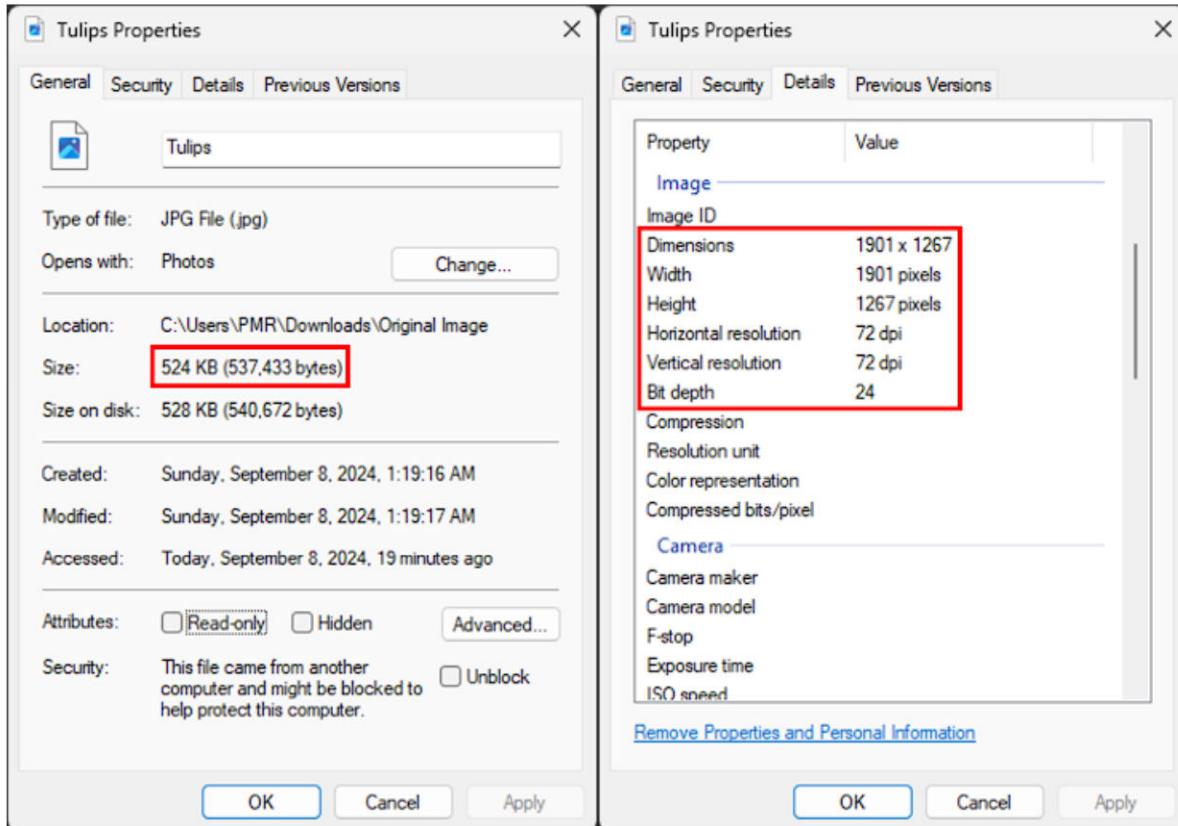
Create a folder named "Original Image" in any directory you prefer. *

Done

Download the tulip image from [Google Drive](#) and save it in the folder. *

Done

Check if the image size and properties match the picture below. *



Done

Open Orange and create a new workflow.

*

Name it in this format: **[Last_Name]_[Section]_Lab_Exercise_3** (e.g., Rey_4A_Lab_Exercise_3.ows).

Done

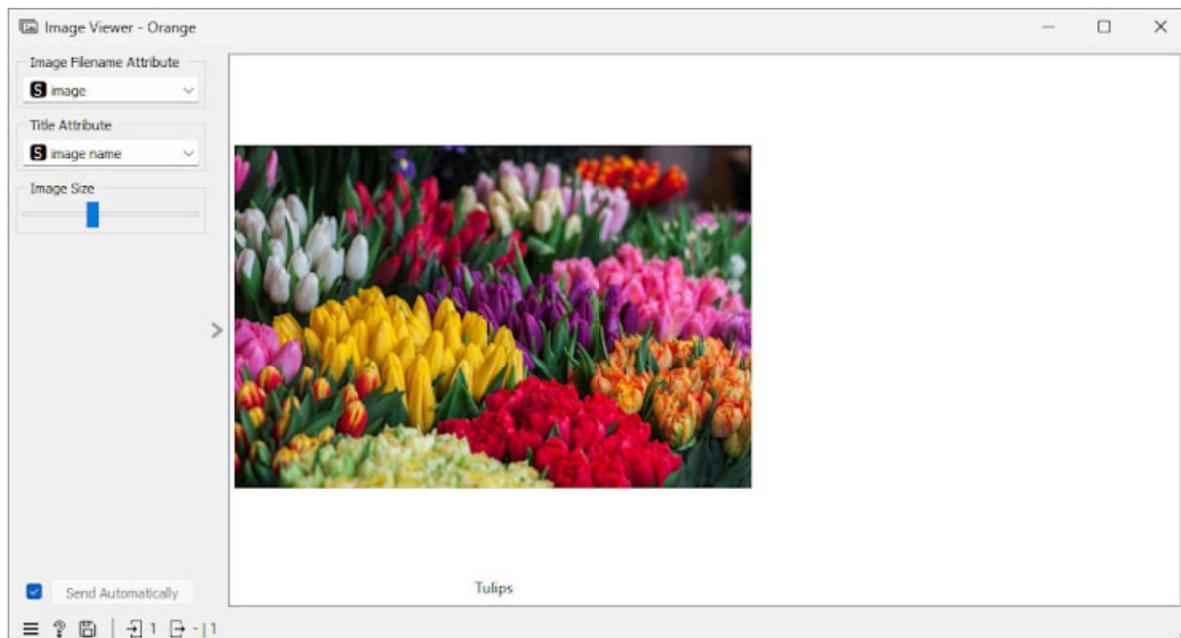
Drag and drop an "Import Image" widget. *

Then, select the "Original Image" folder to load the image.

Done

Use the "Image Viewer" widget connected to "Import Image" to view the image. *

You should see something like this:



Done

Question: If the image width is 1901 pixels and the height is 1267 pixels, what is * the total number of pixels in the image?

Your answer

Download a script named "get_pixels_rgb.py" from [Google Drive](#) and save it in * the "Original Image" folder.

Done

Drag and drop a "Python Script" widget and connect it to "Import Image". *

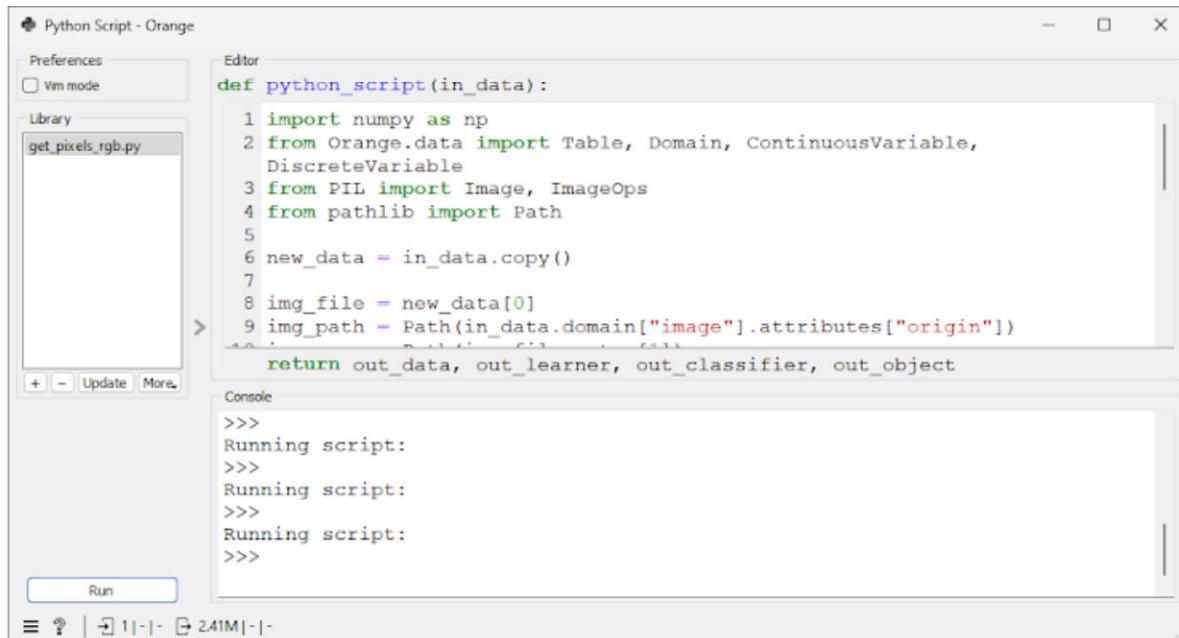
Done

In the "Python Script" window, click the "More" button under the "Library" section * and choose "Import Script from File."

Done

Import the downloaded script and click the "Run" button. This process may take * some time.

You should then see something like this:



The screenshot shows the "Python Script - Orange" application window. On the left, there's a sidebar with "Preferences" (checkbox for "Vim mode"), a "Library" section containing a list box with "get_pixels_rgb.py" selected, and buttons for "+", "-", "Update", and "More...". In the center, the "Editor" pane displays the following Python code:

```
def python_script(in_data):
    1 import numpy as np
    2 from Orange.data import Table, Domain, ContinuousVariable,
        DiscreteVariable
    3 from PIL import Image, ImageOps
    4 from pathlib import Path
    5
    6 new_data = in_data.copy()
    7
    8 img_file = new_data[0]
    9 img_path = Path(in_data.domain["image"].attributes["origin"])
    10
    11 return out_data, out_learner, out_classifier, out_object
```

Below the editor is the "Console" pane, which shows the output of running the script:

```
>>>
Running script:
>>>
Running script:
>>>
Running script:
>>>
```

At the bottom of the window, there's a "Run" button and a status bar showing file paths and sizes.

Done

Analyze the script. *

Question: From lines 17-22, in what order are the RGB values retrieved from the image?

The screenshot shows the Orange data mining software's Python Script interface. The script file 'get_pixels_rgb.py' is open in the editor. The code defines a function 'python_script' that iterates through the pixels of an image to extract RGB values. The relevant lines (17-22) are highlighted with a red box:

```
def python_script(in_data):
    17 data = []
    18 for y in range(height):
    19     for x in range(width):
    20         pixel = img.getpixel((x, y))
    21         r, g, b = pixel
    22         data.append([r, g, b])
    23
    24 data = np.array(data)
    25
    26 domain = Domain([ContinuousVariable("r"), ContinuousVariable("g"),
    27                   ContinuousVariable("b")])
    28 return out_data, out_learner, out_classifier, out_object
```

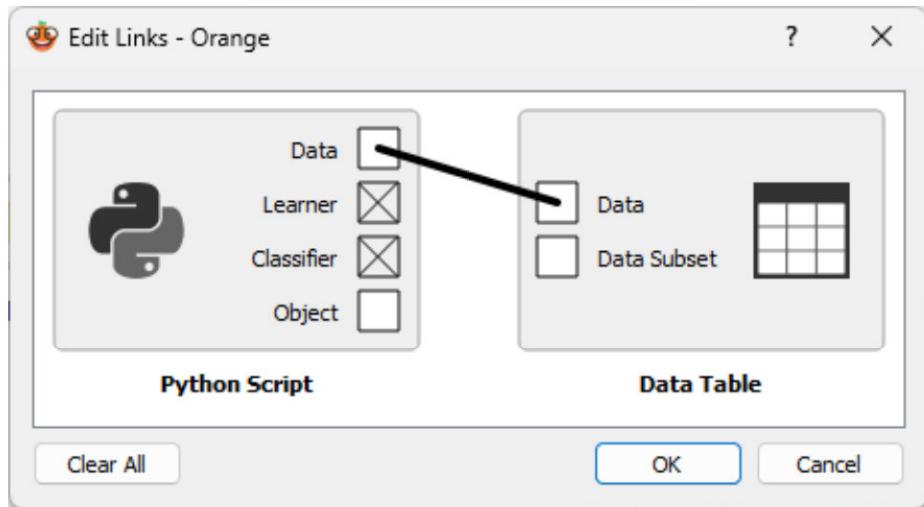
The console window shows the script running, with multiple 'Running script:' messages. A 'Run' button is visible at the bottom left of the editor.

- Moving top to bottom, column by column, from left to right.
- Moving left to right, row by row, from top to bottom.

Drag and drop a "Data Table" widget and connect it to the "Python Script" to get the data that the script outputs. *

- Done

Make sure that "Data Table" is connected to "Python Script" like this: *



- Done

Check the "Info" section in "Data Table". *

Question: What does "2408567 instances" refer to?

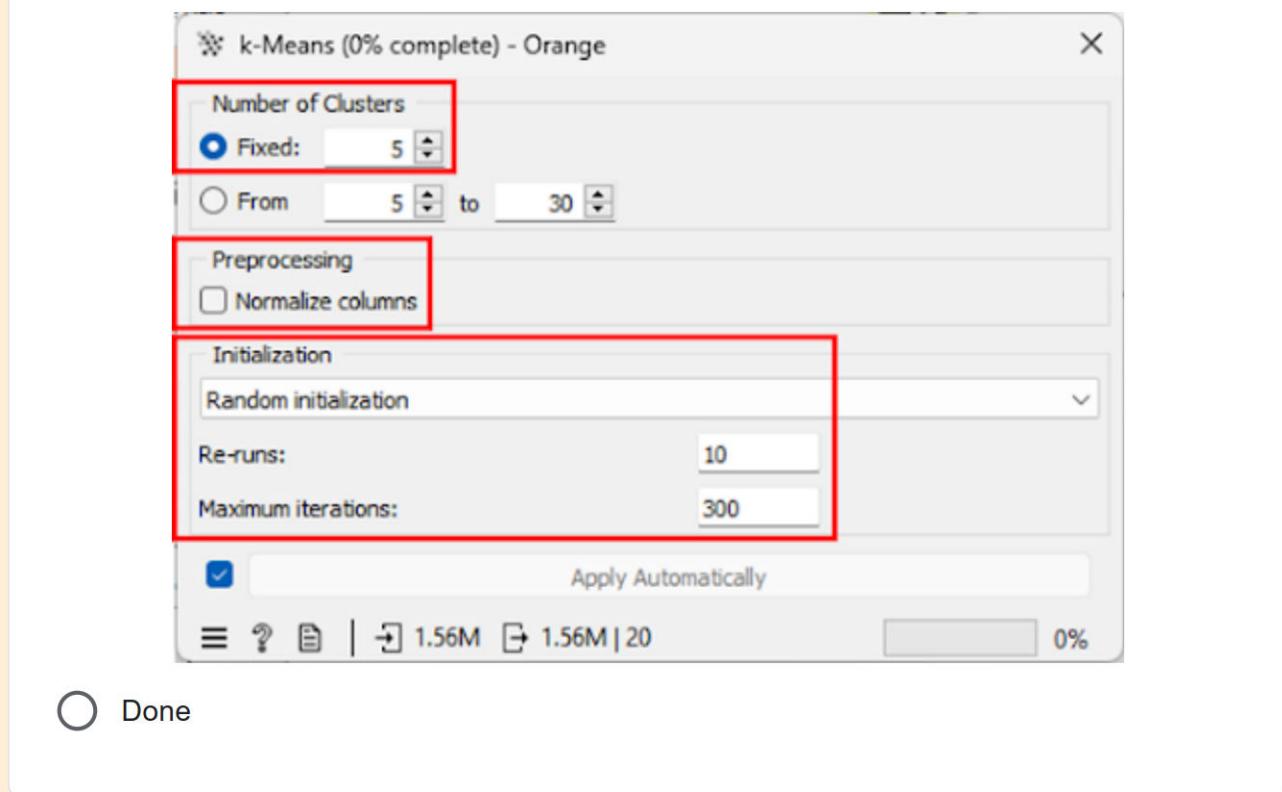
- The total number of pixels in the image.
- The width of the image.
- The height of the image.

Step 2: Apply K-Means to Find the Main Colors

Drag and drop a "k-Means" widget, but do not connect it to "Data Table" yet. *

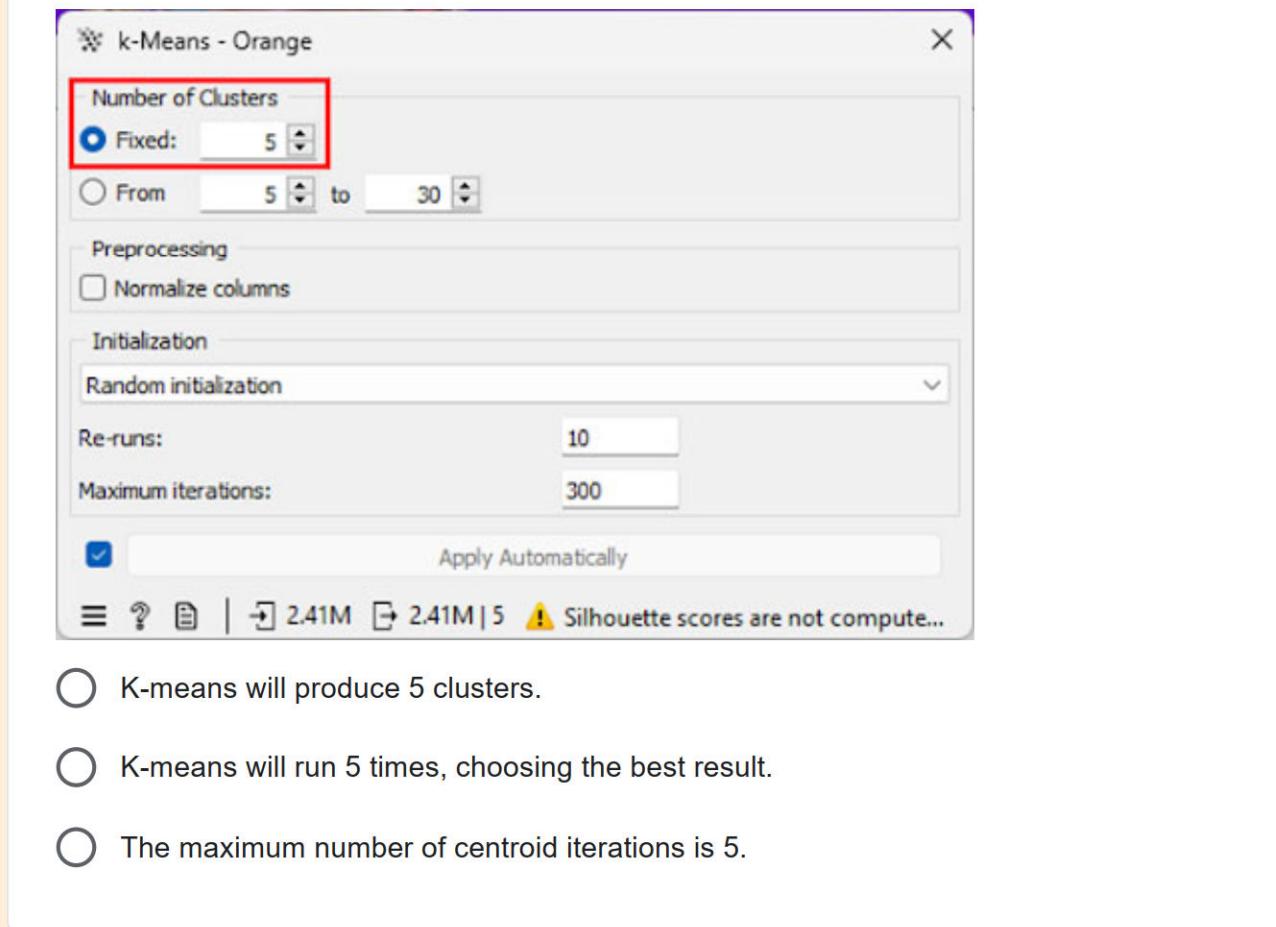
- Done

Set "k-Means" to these settings: *



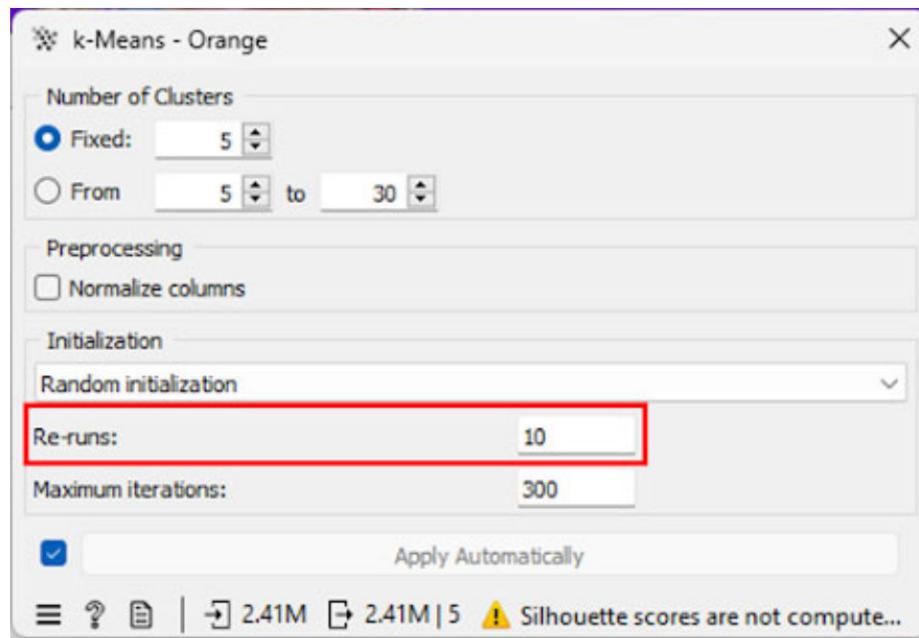
Question: What does the red box in the "k-Means" window image indicate? *

You may refer to this [documentation](#).



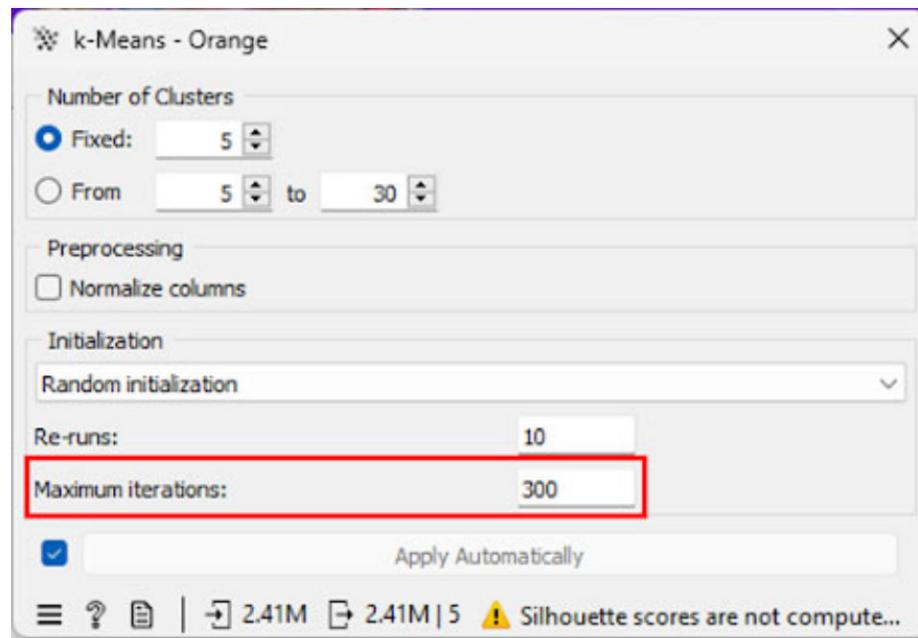
- K-means will produce 5 clusters.
- K-means will run 5 times, choosing the best result.
- The maximum number of centroid iterations is 5.

Question: What does the red box in the "k-Means" window image indicate? *



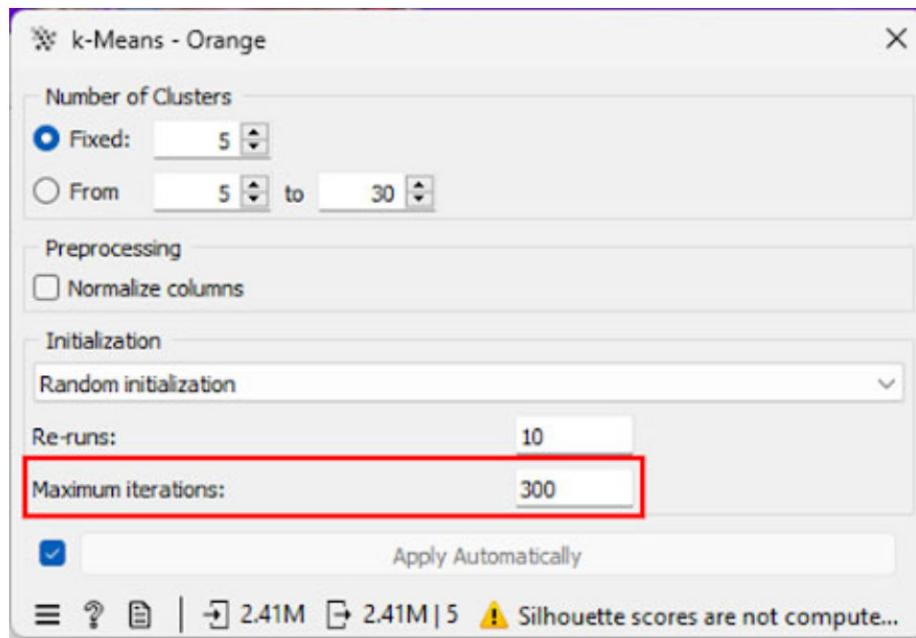
- K-means will run 10 times, choosing the best result.
- K-means will produce 10 clusters.
- The maximum number of centroid iterations is 10.

Question: What does the red box in the "k-Means" window image indicate? *



- The maximum number of centroid iterations is 300.
- K-means will produce 300 clusters.
- K-means will run 300 times, choosing the best result.

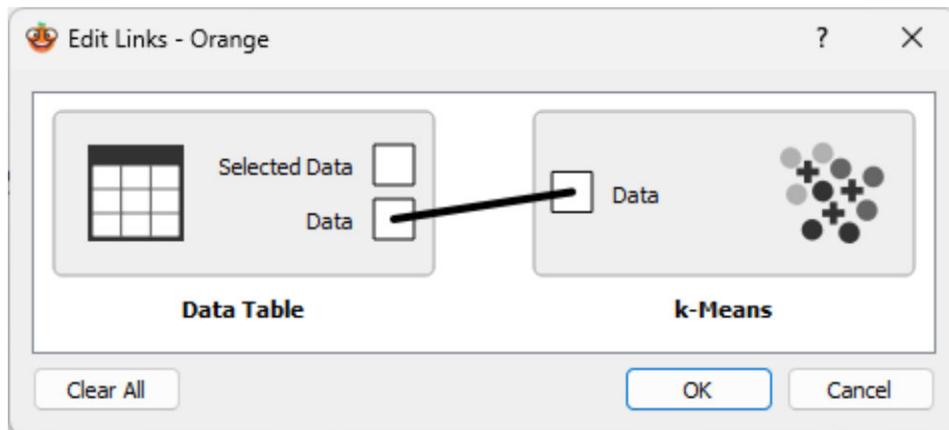
Question: What does the red box in the "k-Means" window image indicate? *



- The maximum number of centroid iterations is 300.
- K-means will produce 300 clusters.
- K-means will run 300 times, choosing the best result.

Connect "k-Means" to "Data Table". *

Make sure it is connected to "Data Table" like this:

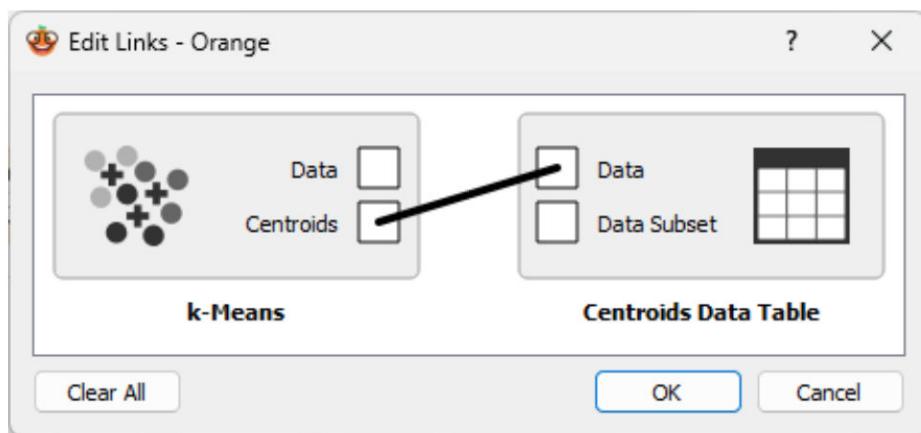


- Done

Drag and drop a "Data Table" widget and name it "Centroids Data Table". *

Done

Make sure "Centroids Data Table" is connected to "k-Means" like this: *

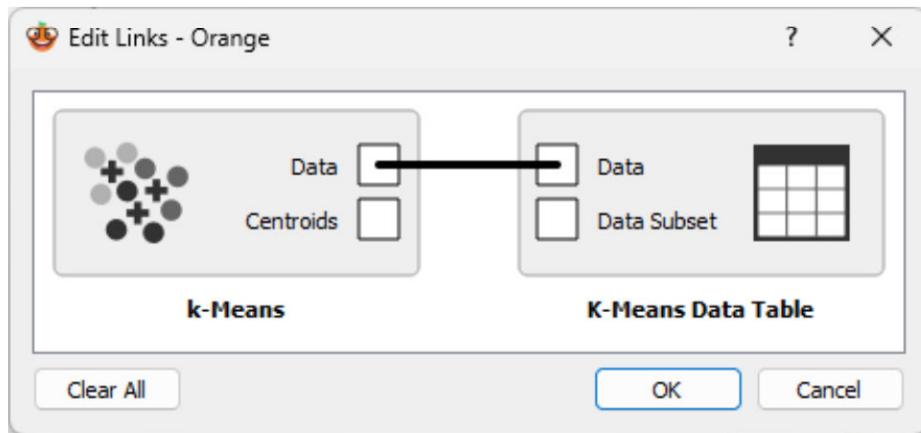


Done

Drag and drop a "Data Table" widget and name it "K-Means Data Table". *

Done

Make sure "K-Means Data Table" is connected to "k-Means" like this: *



Done

Check "Centroids Data Table". *

You should see something like this:

The screenshot shows the 'Centroids Data Table - Orange' window. On the left, there is a configuration panel with the following settings:

- Info**:
 - 5 Instances
 - 3 features
 - No target variable.
 - 2 meta attributes (50.0 % missing data)
- Variables**:
 - Show variable labels (if present) Visualize numeric values
 - Color by instance classes
- Selection**:
 - Select full rows

At the bottom of the configuration panel are two buttons: 'Restore Original Order' and 'Send Automatically' with a checked checkbox.

The main area displays a table with the following data:

	Cluster	Silhouette	r	g	b
1	C1	?	36.7962	35.0312	25.2795
2	C2	?	84.9488	89.8428	69.3953
3	C3	?	173.344	22.0471	49.049
4	C4	?	190.273	151.634	156.626
5	C5	?	206.551	150.502	33.137

Done

Question: Did you get the same RGB values for all 5 clusters? *

If not, try increasing the number of re-runs in "k-Means".

Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137

- Yes
 No

Question: Which color represents Cluster 1? *

Hint: Use an [RGB calculator](#) to find out the color.

Centroids Data Table - Orange

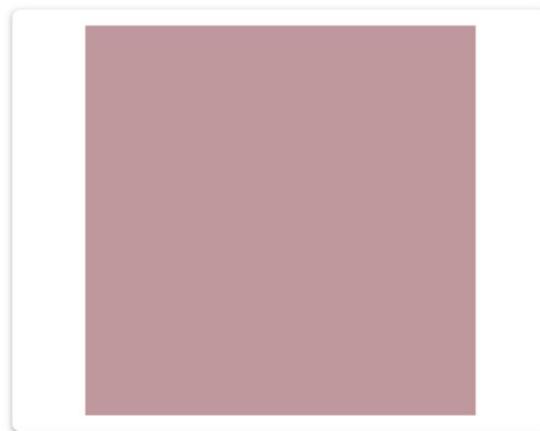
Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137

Info
5 instances
3 features
No target variable.
2 meta attributes (50.0 % missing data)

Variables
 Show variable labels (if present)
 Visualize numeric values
 Color by instance classes

Selection
 Select full rows

Restore Original Order
 Send Automatically



Option 4



Option 2



Option 5



Option 1



Option 3

Question: Which color represents Cluster 2? *

Centroids Data Table - Orange

Info
5 instances
3 features
No target variable.
2 meta attributes (50.0 % missing data)

Variables
 Show variable labels (if present)
 Visualize numeric values
 Color by instance classes

Selection
 Select full rows

Cluster Silhouette r g b

Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137

Restore Original Order Send Automatically

≡ ? ⌂ | ↻ 5 ⌂ 5 | 5



Option 4



Option 1



Option 5



Option 2





Option 3

Question: Which color represents Cluster 3? *

Centroids Data Table - Orange

Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137



Option 3



Option 4



Option 5



Option 2





Option 1

Question: Which color represents Cluster 4? *

Centroids Data Table - Orange

Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137



Option 5



Option 4



Option 2



Option 3





Option 1

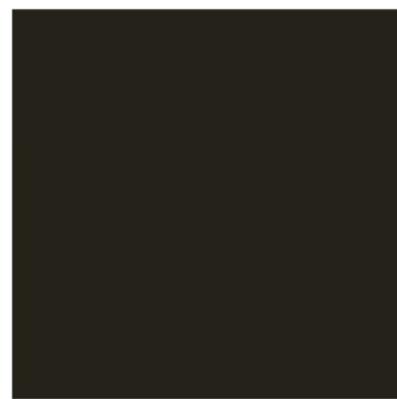
Question: Which color represents Cluster 5? *

Centroids Data Table - Orange

Cluster	Silhouette	r	g	b
1 C1	?	36.7962	35.0312	25.2795
2 C2	?	84.9488	89.8428	69.3953
3 C3	?	173.344	22.0471	49.049
4 C4	?	190.273	151.634	156.626
5 C5	?	206.551	150.502	33.137



Option 2



Option 1



Option 4



Option 5



Option 3

Check "K-Means Data Table". *

You should see something like this. If not, try clicking the "Restore Original Order" button:

The screenshot shows the 'K-Means Data Table - Orange' window. On the left, there is a sidebar with 'Info' (2408567 instances, 3 features, Target with 2 values, 2 meta attributes (50.0 % missing data)), 'Variables' (checkboxes for Show variable labels (if present), Visualize numeric values, Color by instance classes, and Select full rows), and buttons for Restore Original Order and Send Automatically. The main area is a table with columns: Selected, Cluster, Silhouette, r, g, b. The 'Selected' column contains mostly 'No' entries, except for row 1 which is 'Yes'. The 'Cluster' column for all rows is 'C1'. The 'Silhouette' column has question marks. The 'r', 'g', and 'b' columns contain numerical values ranging from 20 to 67.

	Selected	Cluster	Silhouette	r	g	b
1	No	C1	?	28	43	20
2	No	C1	?	30	45	22
3	No	C1	?	34	49	30
4	No	C1	?	38	53	34
5	No	C1	?	36	53	35
6	No	C1	?	35	52	34
7	No	C1	?	35	54	35
8	No	C1	?	37	56	36
9	No	C1	?	34	56	35
10	No	C1	?	34	56	33
11	No	C1	?	34	58	34
12	No	C1	?	34	58	34
13	No	C1	?	33	58	36
14	No	C1	?	35	60	38
15	No	C1	?	39	64	45
16	No	C1	?	40	67	48
17	No	C1	?	35	63	41
18	No	C1	?	36	65	43
19	No	C1	?	32	64	41
20	No	C1	?	35	67	44

Done

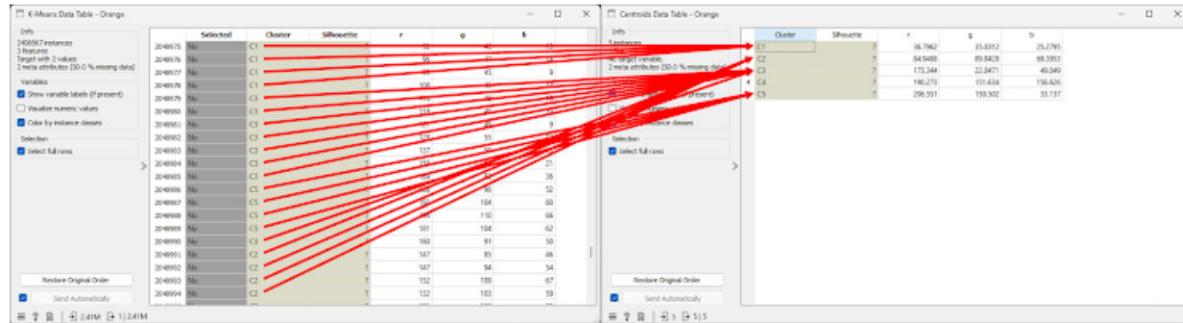
In "K-Means Data Table", each pixel is assigned to a specific cluster.

*

The RGB values for each cluster are stored in a separate data table ("Centroids Data Table").

To determine the new color assigned to each pixel, you need to join the two tables.

Example:

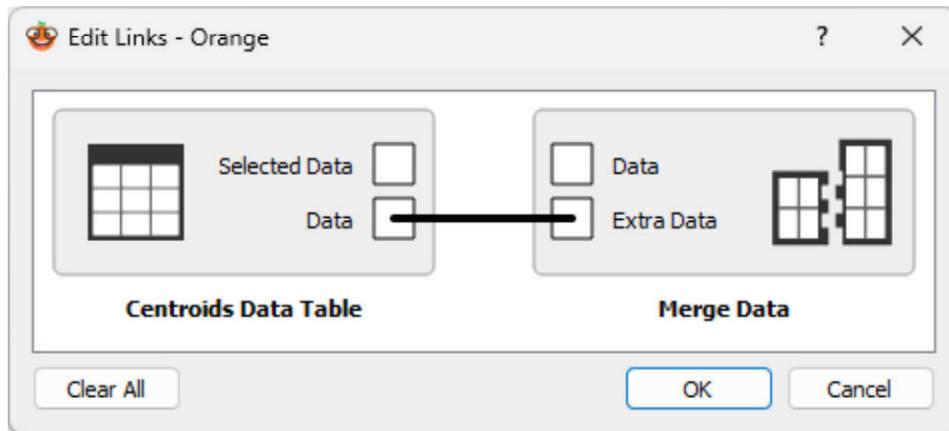


- I have read and understood.

Drag and Drop a "Merge Data" widget. *

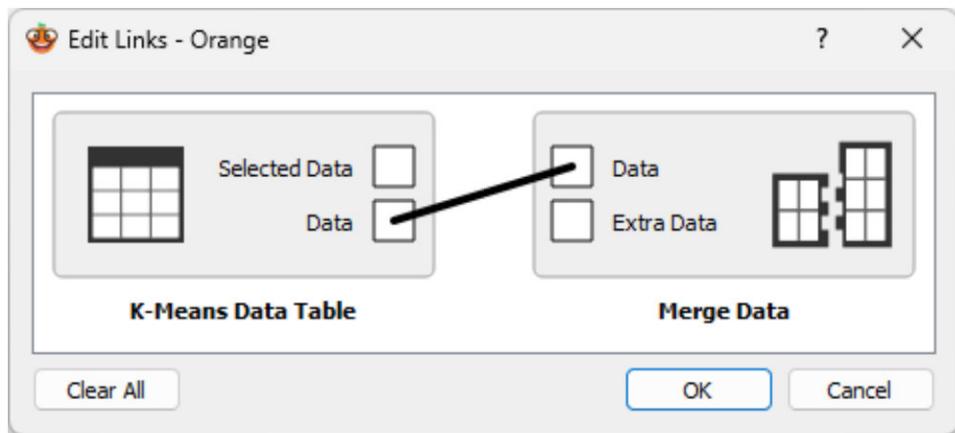
- Done

Make sure "Merge Data" is connected to "Centroids Data Table" like this: *



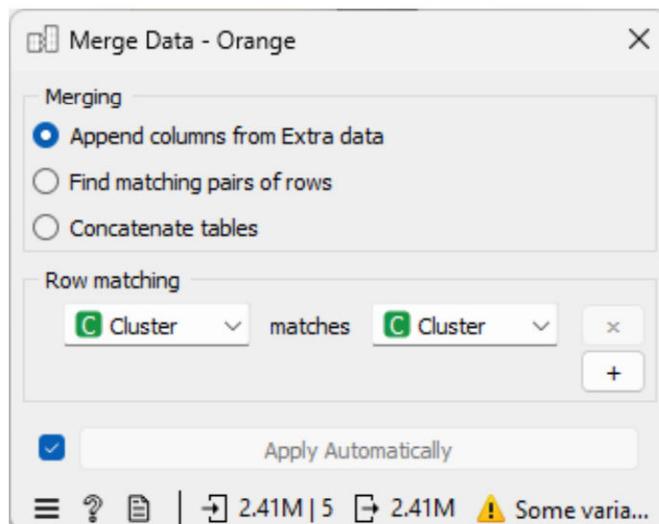
- Done

Make sure "Merge Data" is connected to "K-Means Data Table" like this: *



Done

Set "Merge Data" to these settings: *

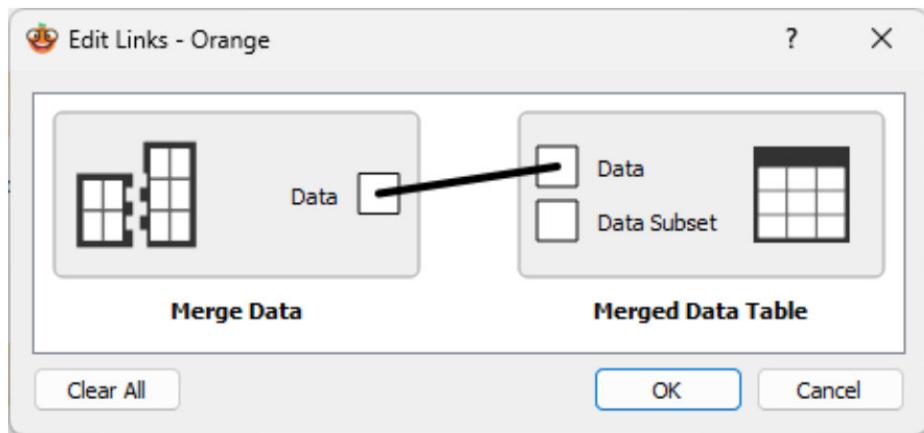


Done

To check if the tables are merged, drag and drop a "Data Table" widget, connect it * to "Merge Data", and name it "Merged Data Table".

Done

Make sure "Merged Data Table" is connected to "Merge Data" like this: *



Done

Check "Merged Data Table". *

You should see something like this. The green box shows the original RGB values for each pixel, while the blue box shows the assigned RGB values based on clustering.

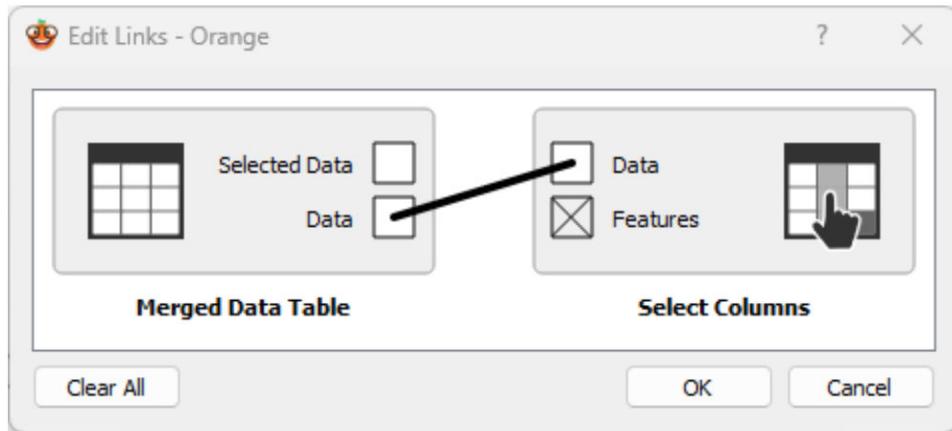
The screenshot shows the 'Merged Data Table - Orange' viewer. The left sidebar contains settings: 'Info' (240856 instances, 6 features, Target with 2 values, 3 meta attributes (33.3 % missing data)), 'Variables' (checkboxes for Show variable labels (if present), Visualize numeric values, Color by instance classes), and 'Selection' (checkbox for Select full rows). The main area displays a table with 21 rows. The first column is labeled 'id (1)' and contains numerical values from 1 to 21. The next three columns ('r (1)', 'g (1)', 'b (1)') show the original RGB values for each pixel. The last three columns ('r (2)', 'g (2)', 'b (2)') show the assigned RGB values based on clustering. A green box highlights the first three columns of data, and a blue box highlights the last three columns of data. The bottom status bar shows 'Send Automatically' checked and file sizes '2.41M' for both the source and target files.

Done

To keep only the updated RGB values for each pixel, drag and drop "Select Columns" and connect it to "Merged Data Table". *

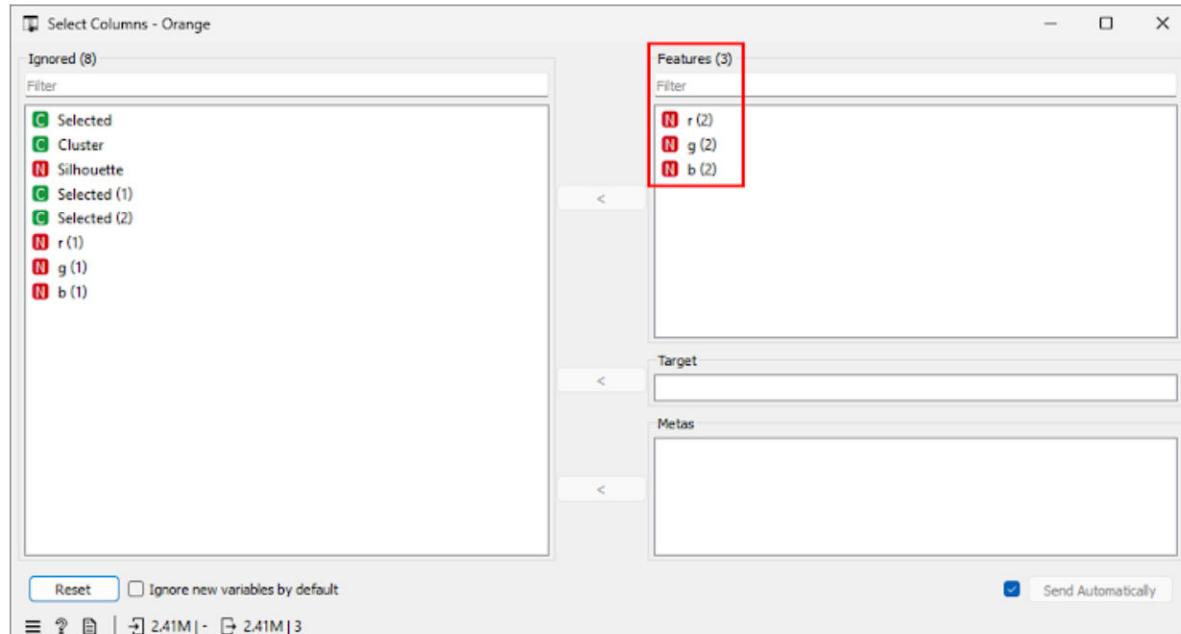
Done

Make sure "Select Columns" is connected to "Merged Data Table" like this: *



Done

Set "Select Columns" as shown in the image to keep only the new RGB values for * each pixel.



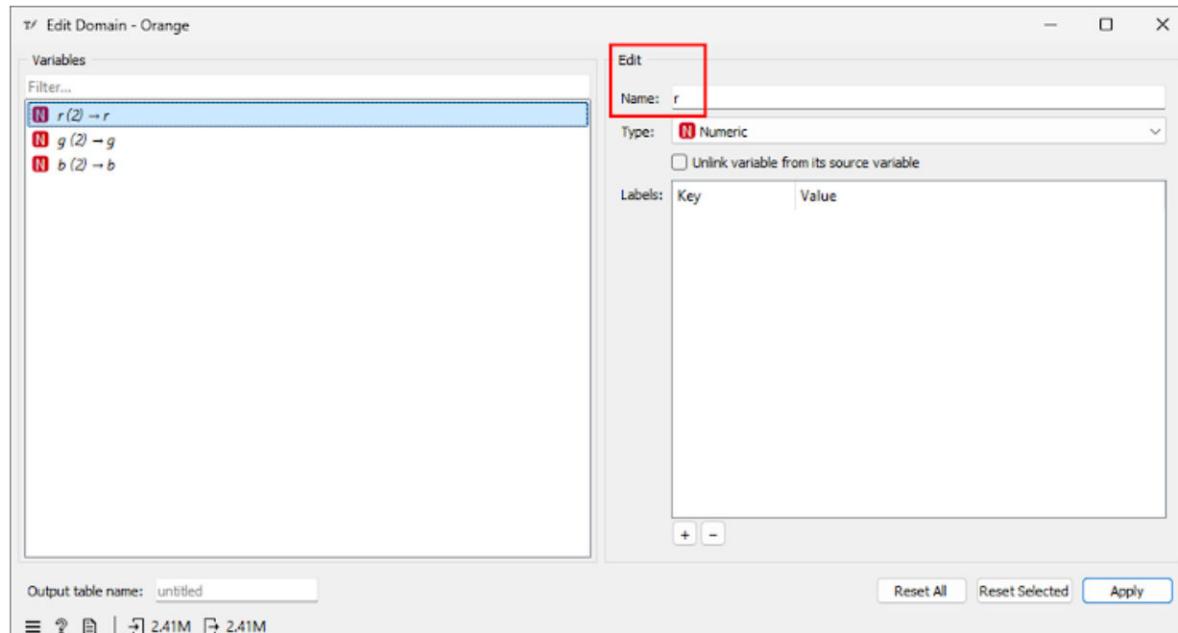
Done

To rename columns (e.g., "r (2)" to "r"), drag and drop an "Edit Domain" widget and connect it to "Select Columns". *

Done

In the "Edit" section of "Edit Domain," change each column name. *

Click the "Apply" button to save the changes.



Done

Step 3: Save the Compressed Image

Pause the workflow using the button in the lower left corner of Orange. *



Done

Download a script named "create_image_from_rgb_values.py" from [Google Drive](#) * and save it in the "Original Image" folder.

Done

Drag and drop another "Python Script" widget. *

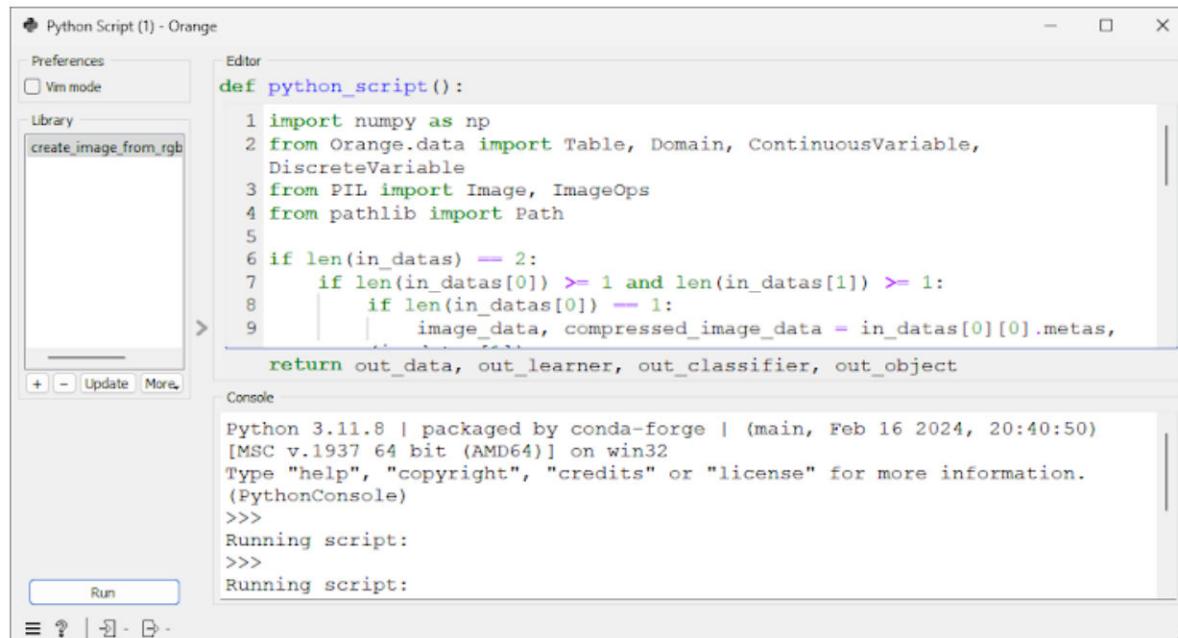
Done

In the "Python Script (1)" window, click the "More" button under the "Library" section and choose "Import Script from File." *

Done

Import the downloaded script. *

You should see something like this:



The screenshot shows the Orange data mining software interface with a Python Script (1) window open. The window has tabs for Preferences, Vim mode, Library, and Editor. In the Library tab, 'create_image_from_rgb' is selected. The Editor tab shows the following Python code:

```
def python_script():
    1 import numpy as np
    2 from Orange.data import Table, Domain, ContinuousVariable,
        DiscreteVariable
    3 from PIL import Image, ImageOps
    4 from pathlib import Path
    5
    6 if len(in_datas) == 2:
    7     if len(in_datas[0]) >= 1 and len(in_datas[1]) >= 1:
    8         image_data, compressed_image_data = in_datas[0][0].metas,
    9
    return out_data, out_learner, out_classifier, out_object
```

The Console tab displays the Python environment and the start of the script execution:

```
Python 3.11.8 | packaged by conda-forge | (main, Feb 16 2024, 20:40:50)
[MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(PythonConsole)
>>>
Running script:
>>>
Running script:
```

Done

Question: What do lines 18-20 do?

The screenshot shows a Python script editor window titled "Python Script (1) - Orange". The code defines a function `python_script()` which takes parameters `out_data`, `out_learner`, `out_classifier`, and `out_object`. Lines 18-20 are highlighted with a red box. The code in these lines reshapes three channels (red, green, blue) from a 1D array into 2D arrays of height and width. Line 21 stacks these 2D arrays along axis -1 to form an RGB image. Line 23 clips the image values to 0-255 and converts them to uint8. Line 25 creates an Image object from the array. The "Console" tab at the bottom shows the Python version and a running script message.

```
def python_script():
    r = compressed_image_data[:, 0].reshape((image_height,
    image_width))
    g = compressed_image_data[:, 1].reshape((image_height,
    image_width))
    b = compressed_image_data[:, 2].reshape((image_height,
    image_width))
    rgb_image = np.stack((r, g, b), axis=-1)
    rgb_image = np.clip(rgb_image, 0, 255).astype(np.uint8)
    image = Image.fromarray(rgb_image, mode='RGB')
    return out_data, out_learner, out_classifier, out_object
```

- Saves the image in the user's "Downloads" folder.
- Extracts the red, green, and blue channels from the table, reshaping each into a 2D array matching the image's height and width in pixels.
- Gets the image's width and height from the metadata.
- Generates an RGB image from the 3D array.
- Combines the red, green, and blue channels into a 3D array.

Question: What does line 22 do?

The screenshot shows a Python script editor window titled "Python Script (1) - Orange". The editor pane contains the following code:

```
def python_script():
    image_width)
    20 |     b = compressed_image_data[:, 2].reshape((image_height,
    image_width))
    21 |
    22 |     rgb_image = np.stack((r, g, b), axis=-1)
    23 |     rgb_image = np.clip(rgb_image, 0, 255).astype(np.uint8)
    24 |
    25 |     image = Image.fromarray(rgb_image, mode='RGB')
    26 |     path = str(Path.home() / "Downloads")
    27 |     image.save(f"(path)\{image_name}_compressed.{image_type}")

    return out_data, out_learner, out_classifier, out_object
```

The line `rgb_image = np.stack((r, g, b), axis=-1)` is highlighted with a red box. The console pane below shows the Python environment and the running of the script:

```
Python 3.11.8 | packaged by conda-forge | (main, Feb 16 2024, 20:40:50)
[MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(PythonConsole)
>>>
Running script:
>>>
Running script:
```

- Combines the red, green, and blue channels into a 3D array.
- Generates an RGB image from the 3D array.
- Gets the image's width and height from the metadata.
- Saves the image in the user's "Downloads" folder.
- Extracts the red, green, and blue channels from the table, reshaping each into a 2D array matching the image's height and width in pixels.

Analyze the script.

*

Question: What does line 25 do?

The screenshot shows a Python script editor window titled "Python Script (1) - Orange". The editor pane contains the following code:

```
def python_script():
    image_width)
    20 |     b = compressed_image_data[:, 2].reshape((image_height,
    image_width))
    21 |
    22 |     rgb_image = np.stack((r, g, b), axis=-1)
    23 |     rgb_image = np.clip(rgb_image, 0, 255).astype(np.uint8)
    24 |
    25 |     image = Image.fromarray(rgb_image, mode='RGB')
    26 |     path = str(Path.home() / "Downloads")
    27 |     image.save(f"(path)\{image_name}_compressed.{image_type}")

    return out_data, out_learner, out_classifier, out_object
```

The line `image = Image.fromarray(rgb_image, mode='RGB')` is highlighted with a red box. The console pane below shows the Python environment and the running of the script:

```
Python 3.11.8 | packaged by conda-forge | (main, Feb 16 2024, 20:40:50)
[MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(PythonConsole)
>>>
Running script:
>>>
Running script:
```

- Extracts the red, green, and blue channels from the table, reshaping each into a 2D array matching the image's height and width in pixels.
- Generates an RGB image from the 3D array.
- Saves the image in the user's "Downloads" folder.
- Gets the image's width and height from the metadata.
- Combines the red, green, and blue channels into a 3D array.

Analyze the script.

*

Question: What does line 26 do?

The screenshot shows the Python Script (1) window in the Orange data mining interface. The code editor contains the following Python script:

```
def python_script():
    image_width)
    20 |     b = compressed_image_data[:, 2].reshape((image_height,
    image_width))
    21 |
    22 |     rgb_image = np.stack((r, g, b), axis=-1)
    23 |     rgb_image = np.clip(rgb_image, 0, 255).astype(np.uint8)
    24 |
    25 |     image = Image.fromarray(rgb_image, mode='RGB')
    26 |     path = str(Path.home() / "Downloads")
    27 |     image.save(f"(path)\{image_name}_compressed.{image_type}")

    return out_data, out_learner, out_classifier, out_object
```

The line `path = str(Path.home() / "Downloads")` is highlighted with a red box. The console output below shows the Python environment and the start of the script execution.

- Extracts the red, green, and blue channels from the table, reshaping each into a 2D array matching the image's height and width in pixels.
- Generates an RGB image from the 3D array.
- Combines the red, green, and blue channels into a 3D array.
- Gets the image's width and height from the metadata.
- Saves the image in the user's "Downloads" folder.

Connect "Python Script (1)" to "Edit Domain" and "Import Images". *

- Done

To resume the workflow, click the "Pause" button again. *

- Done

Check "Python Script (1)".

*

Click the "Run" button to execute the downloaded script. This process may take some time.

Done

Check your "Downloads" folder for an image named "Tulips_compressed.jpg". *

Done

Rename it to "k5.jpg" to show that this image was created using five clusters. *

Done

Create a folder named "Compressed Image" and move the compressed image into it. *

Done

Question: What does "k5.jpg" look like? *



Option 1



Option 4



Option 2



Option 3

Generate images using 10, 20, and 30 clusters by adjusting "Number of Clusters" * in "k-Means".

Rename each image before changing the number of clusters again.

Done

Open the images with 5, 10, 20, and 30 clusters in Photo Viewer at the same time * to see both the photos and their sizes.

Done

Check the file size of the original image.

*

If it is larger than the compressed images, k-Means can effectively reduce the size of this image.

If not, review and adjust your workflow.

- I have read and understood.

Check the file sizes of the four compressed images.

*

If the file size decreases with fewer clusters, it indicates that less detail leads to a smaller file size for this image.

If not, review and adjust your workflow.

- I have read and understood.

Use the 2x2 Snap Layout to view all four images at once.

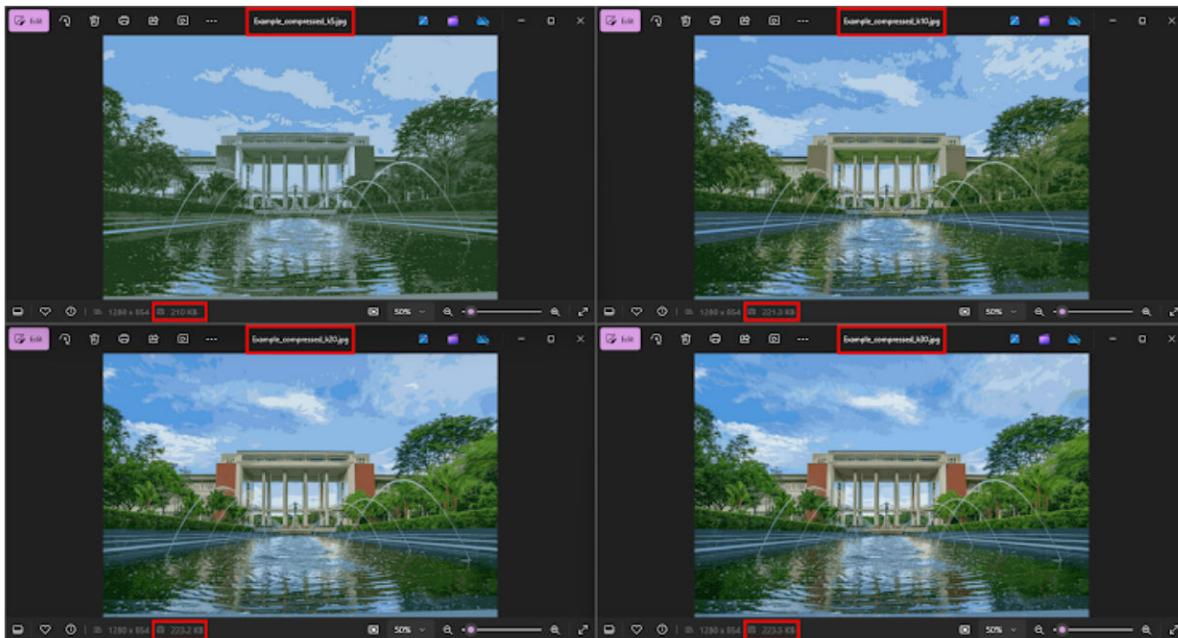
*

Take a full screenshot with the following arrangement:

- Top-left: 5 clusters
- Top-right: 10 clusters
- Bottom-left: 20 clusters
- Bottom-right: 30 clusters

It should display the file names along with their corresponding cluster and file sizes.

Example:



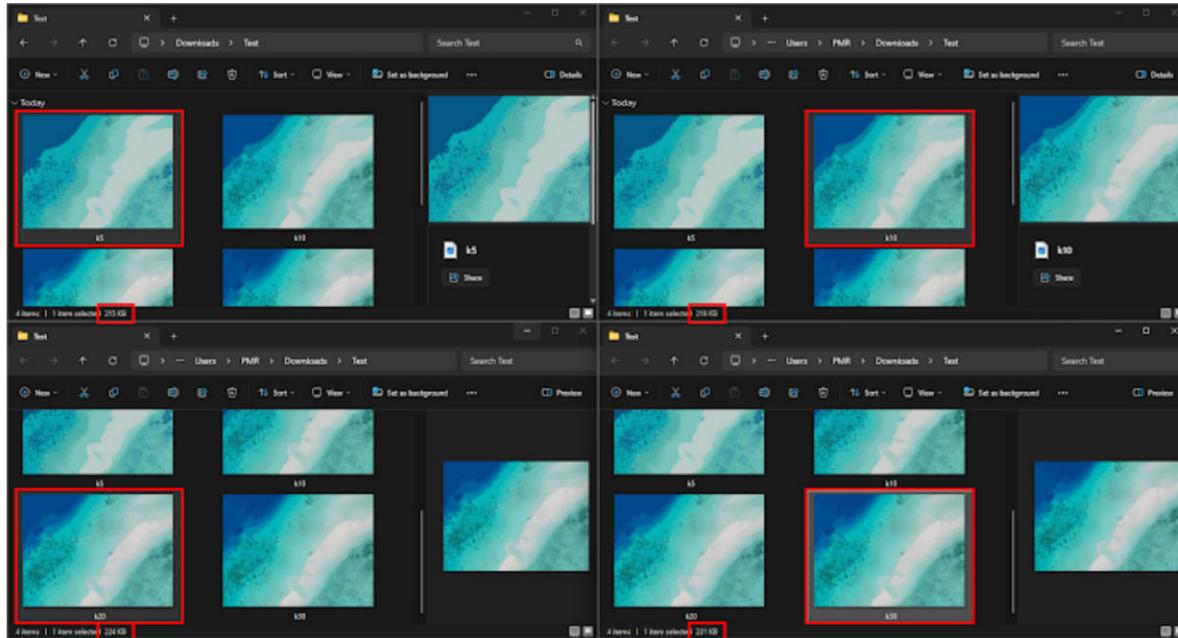
Done

You can also try using File Explorer.

*

It should display the file names along with their corresponding cluster and file sizes.

Example:



Done

Upload the screenshot here, showing that file sizes decrease with fewer clusters and are smaller than the original.

*

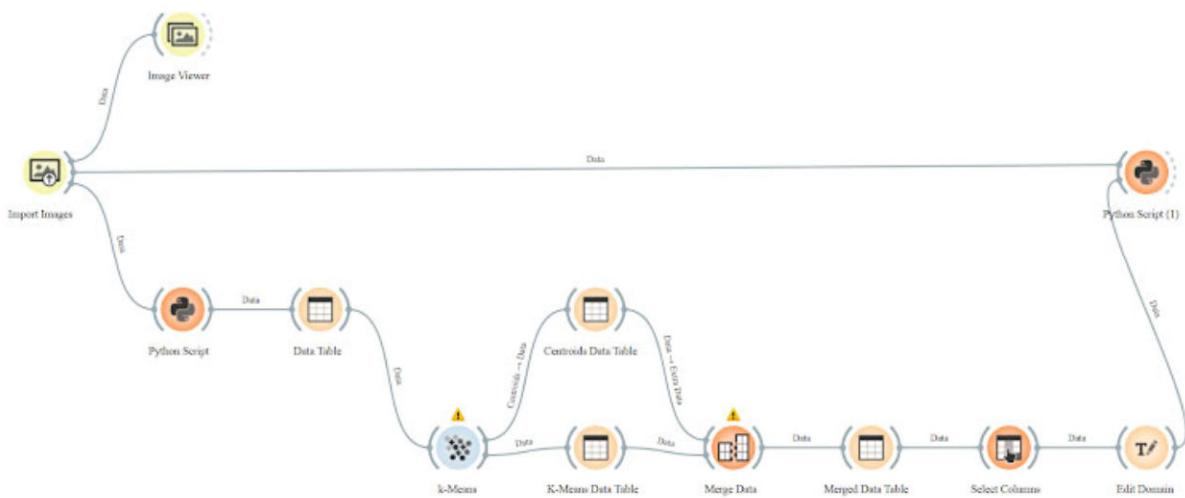
[Add file](#)

Step 4: Save your Workflow

Compare your workflow to this.

*

If they differ, review and adjust your workflow.



I have the same workflow.

Save your workflow and upload it here. *

No need to ZIP the file.

Step 5: Test it Yourself

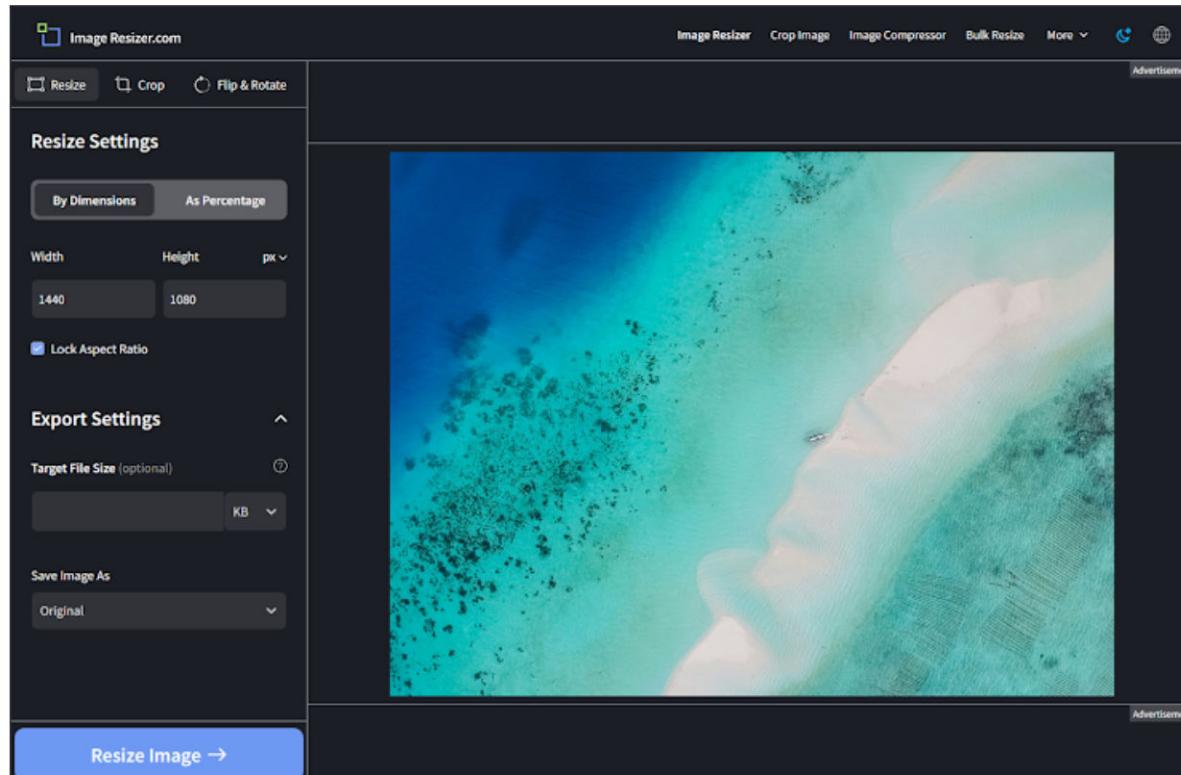
Download a photo of your choice from [Unsplash](#). *

Done

To prevent computer crashes and for demonstration purposes, resize the image * using this [website](#).

Set the height to 1080px and lock the aspect ratio.

Example:



Done

Download the resized image and save it to your "Original Image" folder. *

Keep only one image in the folder; delete any others.

Done

Repeat the process to generate images using 5, 10, 20, and 30 clusters. *

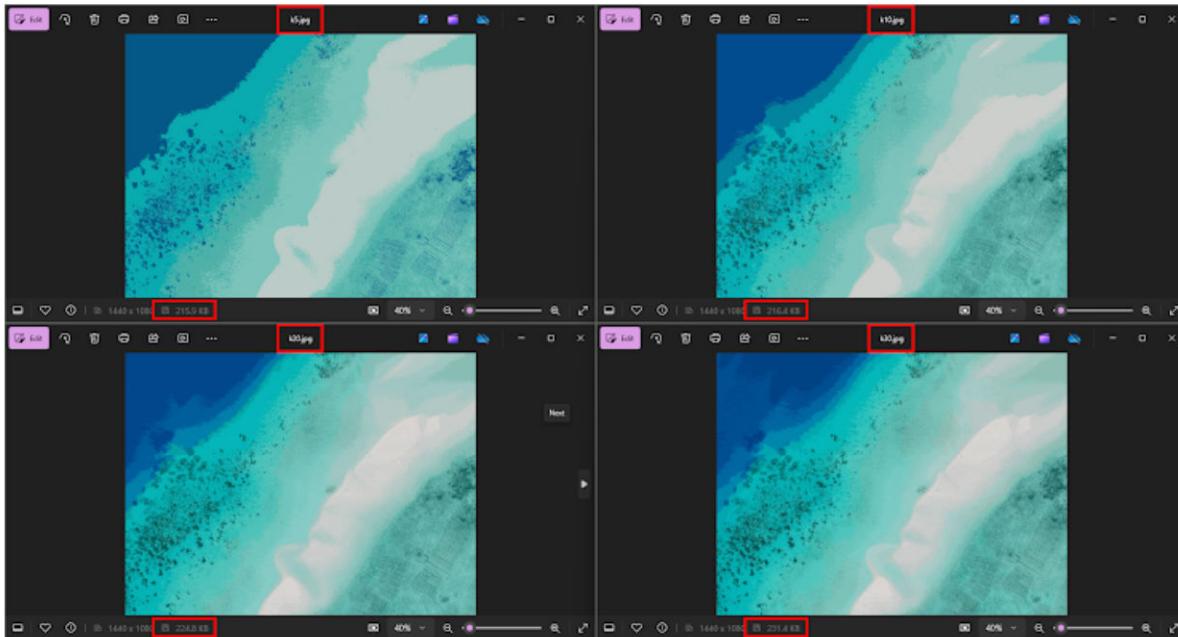
Done

Use the 2x2 Snap Layout to view all four images at once. *

Take a full screenshot with the following arrangement:

- Top-left: 5 clusters
- Top-right: 10 clusters
- Bottom-left: 20 clusters
- Bottom-right: 30 clusters

Example:



Done

Check if the file sizes decrease with fewer clusters and are smaller than the original. *

If not, review and adjust your workflow, or try a different photo.

The file sizes decrease with fewer clusters and are smaller than the original.

Upload the screenshot here, **showing that file sizes decrease with fewer clusters and are smaller than the original.** *

[Add file](#)

[Get link](#)

Never submit passwords through Google Forms.

This form was created inside of Marinduque State College.
Does this form look suspicious? [Report](#)

Google Forms

Content Curated by Pollux M. Rey