Data Mining Homework: Clustering - AI Assistance Log

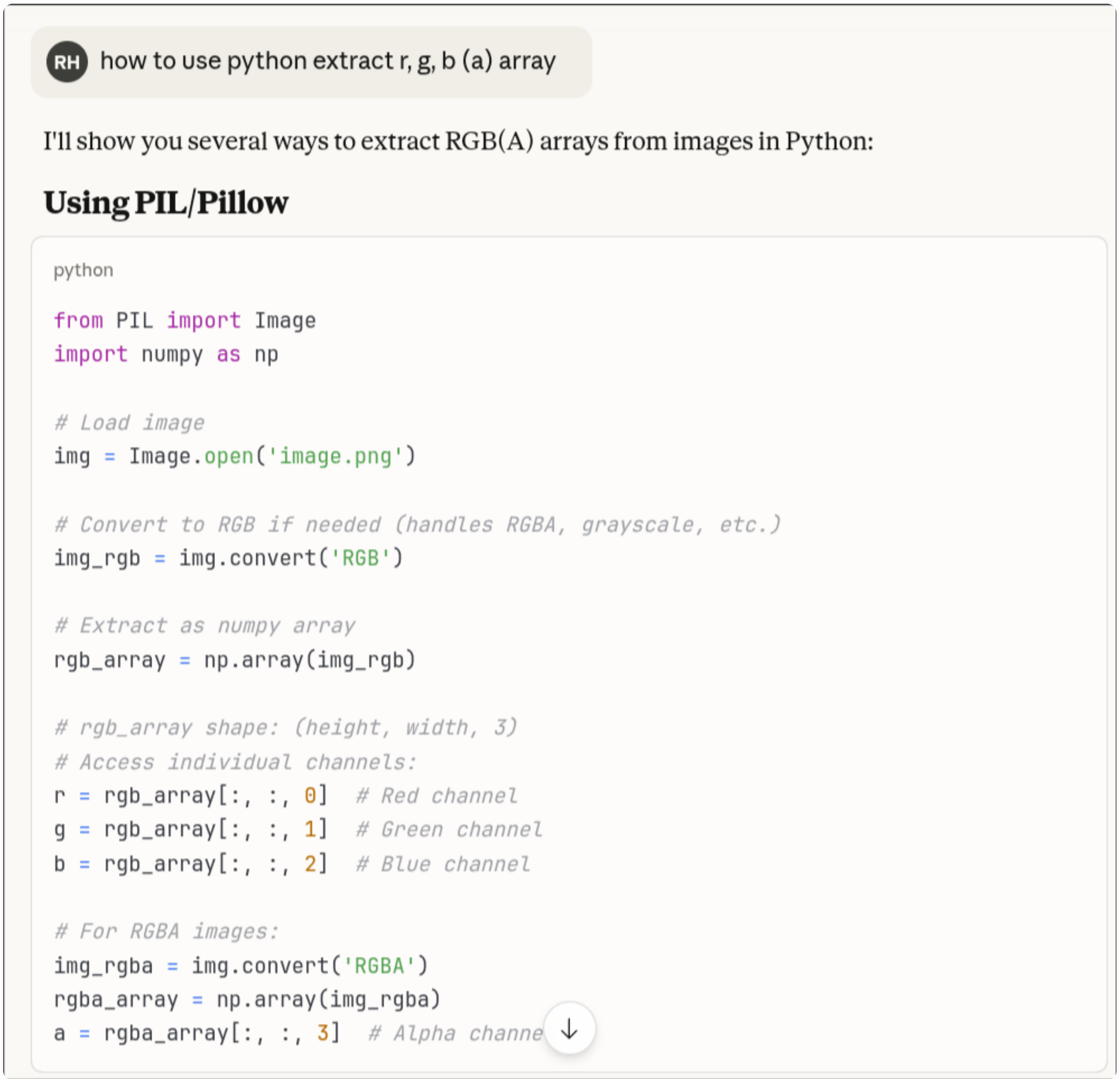
This document records the questions I asked AI and the assistance I received while completing this project.

Question List

During the development of this project, I asked AI the following four core questions:

1. How to extract RGB information from an image
2. How to iterate through that information
3. How to use the K-means algorithm in scikit-learn
4. Help me write a function to compare different K values and display the results

Questions 1 and 2: Image Loading and Data Processing



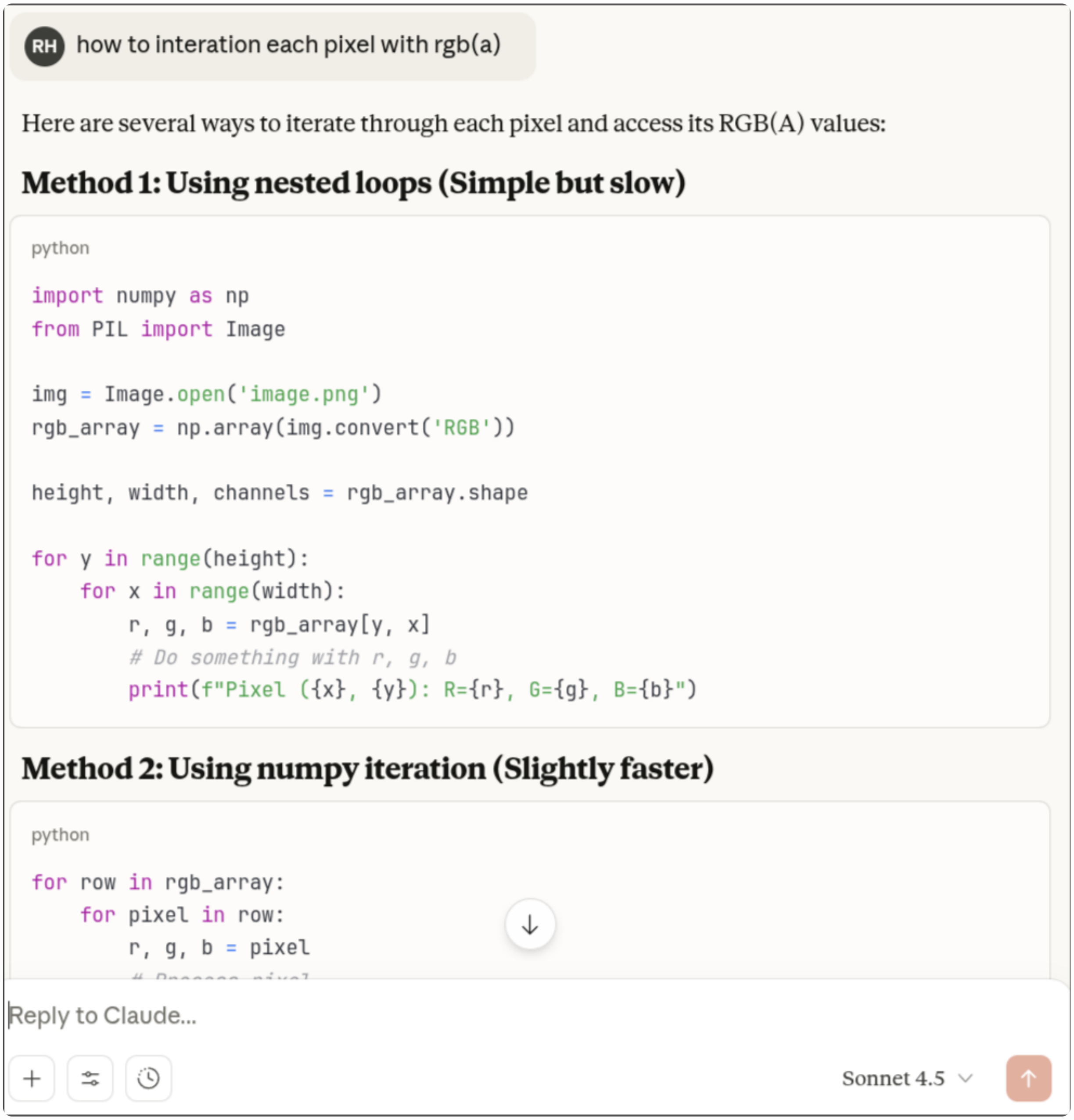
RH how to use python extract r, g, b (a) array

I'll show you several ways to extract RGB(A) arrays from images in Python:

Using PIL/Pillow

python

```python
from PIL import Image
import numpy as np

# Load image
img = Image.open('image.png')

# Convert to RGB if needed (handles RGBA, grayscale, etc.)
img_rgb = img.convert('RGB')

# Extract as numpy array
rgb_array = np.array(img_rgb)

# rgb_array shape: (height, width, 3)
# Access individual channels:
r = rgb_array[:, :, 0]  # Red channel
g = rgb_array[:, :, 1]  # Green channel
b = rgb_array[:, :, 2]  # Blue channel

# For RGBA images:
img_rgba = img.convert('RGBA')
rgba_array = np.array(img_rgba)
a = rgba_array[:, :, 3]  # Alpha channe
```



how to interation each pixel with rgb(a)

Here are several ways to iterate through each pixel and access its RGB(A) values:

Method 1: Using nested loops (Simple but slow)

```python
import numpy as np
from PIL import Image

img = Image.open('image.png')
rgb_array = np.array(img.convert('RGB'))

height, width, channels = rgb_array.shape

for y in range(height):
    for x in range(width):
        r, g, b = rgb_array[y, x]
        # Do something with r, g, b
        print(f"Pixel ({x}, {y}): R={r}, G={g}, B={b}")
```

Method 2: Using numpy iteration (Slightly faster)

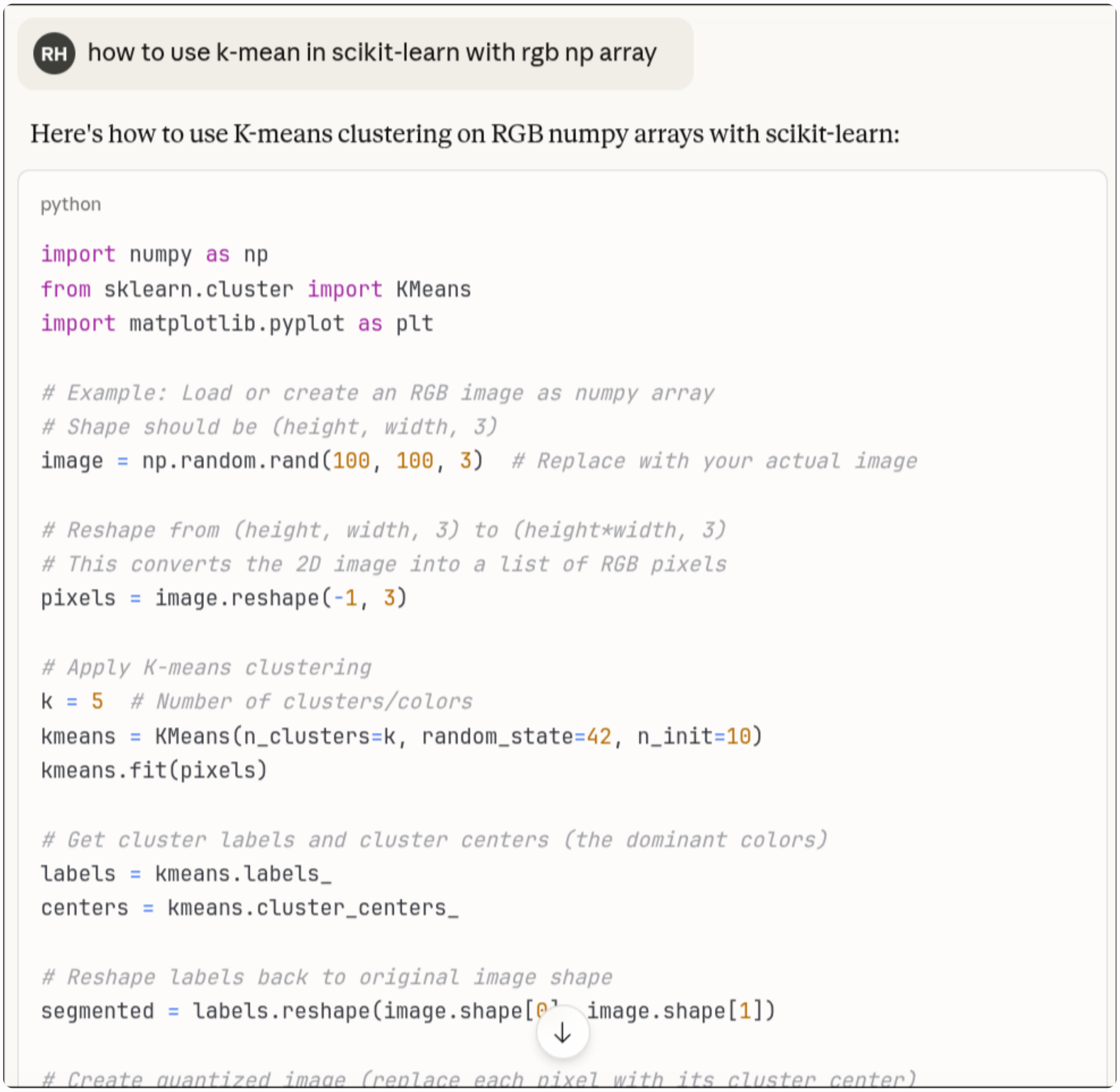
```python
for row in rgb_array:
    for pixel in row:
        r, g, b = pixel
```

Through the discussion of the first two questions, I obtained the following code:

```
def load_image(image_path):
    """載入圖片並轉換為 numpy 陣列"""
    img = Image.open(image_path)
    img_array = np.array(img)
    return img_array
```

This function is responsible for reading image files and converting them into numpy array format for subsequent processing.

Question 3: K-means Image Compression Implementation



how to use k-mean in scikit-learn with rgb np array

Here's how to use K-means clustering on RGB numpy arrays with scikit-learn:

```python
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Example: Load or create an RGB image as numpy array
# Shape should be (height, width, 3)
image = np.random.rand(100, 100, 3)  # Replace with your actual image

# Reshape from (height, width, 3) to (height*width, 3)
# This converts the 2D image into a list of RGB pixels
pixels = image.reshape(-1, 3)

# Apply K-means clustering
k = 5  # Number of clusters/colors
kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
kmeans.fit(pixels)

# Get cluster labels and cluster centers (the dominant colors)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Reshape labels back to original image shape
segmented = labels.reshape(image.shape[0], image.shape[1])

# Create quantized image (replace each pixel with its cluster center)
```

Based on the answer to the third question, I obtained the core image compression function:

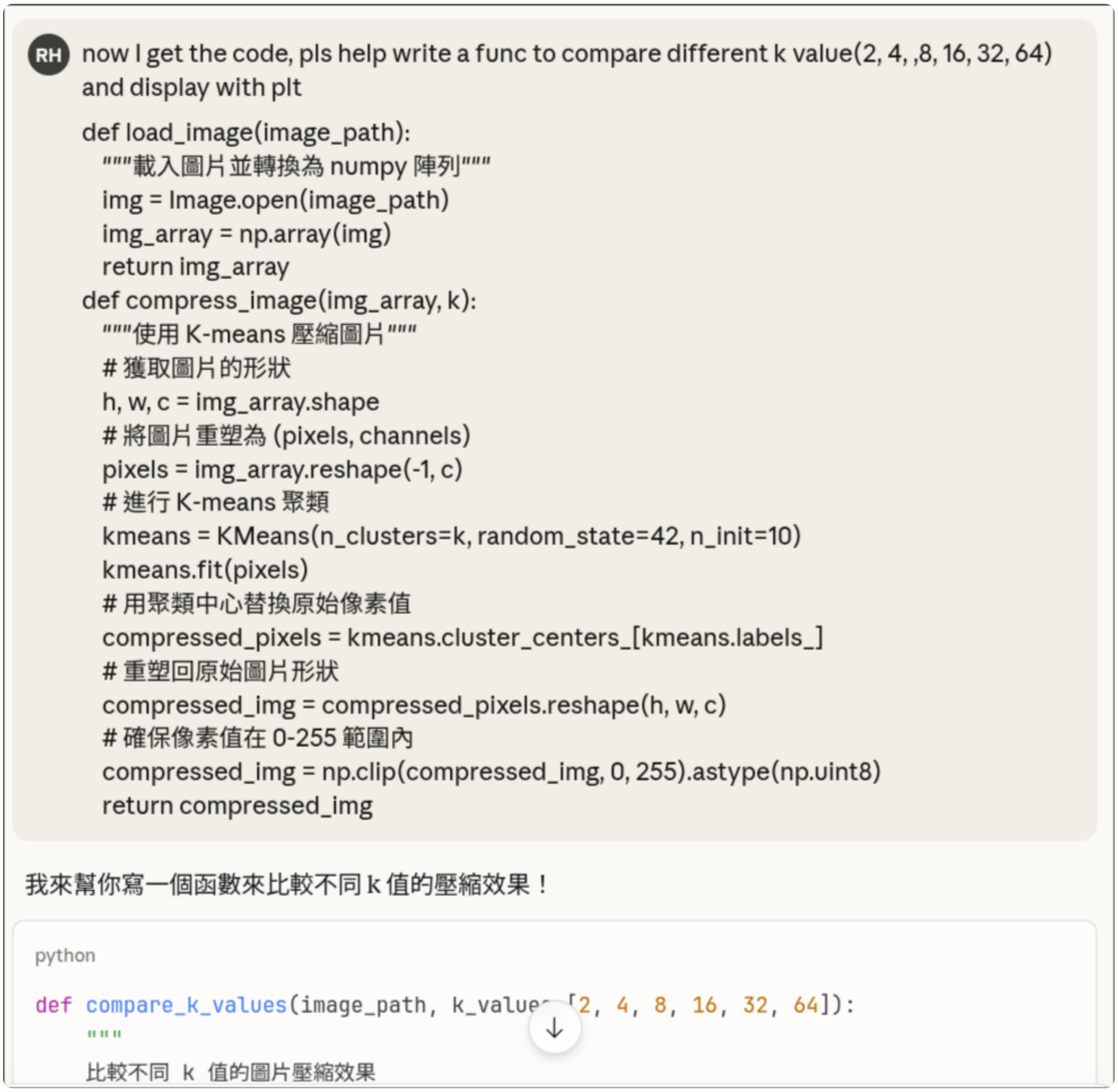
```python
def compress_image(img_array, k):
    """使用 K-means 壓縮圖片"""
    # 獲取圖片的形狀
    h, w, c = img_array.shape
    # 將圖片重塑為 (pixels, channels)
    pixels = img_array.reshape(-1, c)
    # 進行 K-means 聚類
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pixels)
    # 用聚類中心替換原始像素值
    compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
    # 重塑回原始圖片形狀
    compressed_img = compressed_pixels.reshape(h, w, c)
    # 確保像素值在 0-255 範圍內
    compressed_img = np.clip(compressed_img, 0, 255).astype(np.uint8)
    return compressed_img
```

Core Concept Explanation:

The two functions above are the core components of this program:

- `load_image()` handles data input
- `compress_image()` implements the K-means clustering algorithm to compress image colors

Question 4: K-Value Comparison and Visualization



now I get the code, pls help write a func to compare different k value(2, 4, ,8, 16, 32, 64) and display with plt

```
def load_image(image_path):
  """載入圖片並轉換為 numpy 陣列"""
  img = Image.open(image_path)
  img_array = np.array(img)
  return img_array
def compress_image(img_array, k):
  """使用 K-means 壓縮圖片"""
  # 獲取圖片的形狀
  h, w, c = img_array.shape
  # 將圖片重塑為 (pixels, channels)
  pixels = img_array.reshape(-1, c)
  # 進行 K-means 聚類
  kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
  kmeans.fit(pixels)
  # 用聚類中心替換原始像素值
  compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
  # 重塑回原始圖片形狀
  compressed_img = compressed_pixels.reshape(h, w, c)
  # 確保像素值在 0-255 範圍內
  compressed_img = np.clip(compressed_img, 0, 255).astype(np.uint8)
  return compressed_img
```

我來幫你寫一個函數來比較不同 k 值的壓縮效果！

```python
def compare_k_values(image_path, k_value[2, 4, 8, 16, 32, 64]):
    """
    比較不同 k 值的圖片壓縮效果
```

Finally, I asked AI to help me write a function to compare the effects of different K values and display the results:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from PIL import Image
```

```python
import os
def load_image(image_path):
    """載入圖片並轉換為 numpy 陣列"""
    img = Image.open(image_path)
    img_array = np.array(img)
    return img_array
def compress_image(img_array, k):
    """使用 K-means 壓縮圖片"""
    # 獲取圖片的形狀
    h, w, c = img_array.shape
    # 將圖片重塑為 (pixels, channels)
    pixels = img_array.reshape(-1, c)
    # 進行 K-means 聚類
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(pixels)
    # 用聚類中心替換原始像素值
    compressed_pixels = kmeans.cluster_centers_[kmeans.labels_]
    # 重塑回原始圖片形狀
    compressed_img = compressed_pixels.reshape(h, w, c)
    # 確保像素值在 0-255 範圍內
    compressed_img = np.clip(compressed_img, 0, 255).astype(np.uint8)
    return compressed_img
def compare_k_values(image_path, k_values):
    """比較不同 k 值的壓縮結果"""
    # 載入原始圖片
    original_img = load_image(image_path)
    h, w, c = original_img.shape
    # 計算原始圖片的顏色數量
    pixels = original_img.reshape(-1, c)
    original_colors = len(np.unique(pixels, axis=0))
    # 設置子圖
    n_images = len(k_values) + 1
    cols = 3
    rows = (n_images + cols - 1) // cols
    fig, axes = plt.subplots(rows, cols, figsize=(15, 5 * rows))
    axes = axes.flatten() if n_images > 1 else [axes]
    # 顯示原始圖片
    axes[0].imshow(original_img)
    axes[0].set_title(
        f"original image\ncount of colors: {original_colors}\nsize: {h}x{w}",
        fontsize=10,
    )
    axes[0].axis("off")
    # 對每個 k 值進行壓縮並顯示
    for idx, k in enumerate(k_values, 1):
        print(f"正在處理 k={k}...")
        compressed_img = compress_image(original_img, k)
        axes[idx].imshow(compressed_img)
        axes[idx].set_title(f"K = {k}\ncount of colors: {k}", fontsize=10)
        axes[idx].axis("off")
    # 隱藏多餘的子圖
    for idx in range(n_images, len(axes)):
        axes[idx].axis("off")
    plt.tight_layout()
    plt.savefig("kmeans_compression_comparison.png", dpi=150, bbox_inches="tight"
    print("\n比較圖已保存為 'kmeans_compression_comparison.png'")
    plt.show()
def main():
    # 使用 obama.webp 圖片
    image_path = "obama.webp"
    if not os.path.exists(image_path):
        print(f"錯誤: 未找到圖片 '{image_path}'")
        return
    # 要比較的 k 值列表
    k_values = [2, 4, 8, 16, 32, 64]
    print(f"\n開始 K-means 圖片壓縮比較")
    print(f"圖片路徑: {image_path}")
    print(f"K 值: {k_values}\n")
    # 執行比較
    compare_k_values(image_path, k_values)
if __name__ == "__main__":
    main()
```

Summary

Through interactions with AI, I successfully built a complete K-means image compression program. The program is capable of:

- Loading and processing image data
- Using the K-means algorithm for color clustering
- Comparing the compression effects of different K values
- Visualizing the compression results