The Color Signature: Analyzing Art with K-Means clustering

Paloma L. Sett
(PUC-Rio)

1. Introduction

What makes a Van Gogh painting feel like a Van Gogh? Beyond the iconic brushstrokes, a huge part of any artist's identity is their unique use of color. An artist's palette is not just a technical choice; it is the very soul of their work, conveying emotion, atmosphere, and meaning. But can we analyze this soul with the objective tools of science?

The most common use for AI in the creative world today is content *generation*. However, a deeper, less-explored application exists: using AI not as a creator, but as an instrument of insight, a new lens to understand and appreciate human art.

In this article, it will demonstrate a practical use case for how a simple machine learning algorithm can act as a bridge between art and science. It will build a Python tool to extract the quantitative "Color Signature" from any painting and explore what this data reveals about the emotional tension in some of the world's most famous masterpieces. This is not about replacing the art critic with an algorithm, but about equipping our curiosity with a new kind of microscope.

2. The Toolkit: Libraries and Setup

To begin the analysis, it will need a few essential Python libraries, all of which are staples in the data science and computer vision fields. The toolkit is simple and powerful:

- **OpenCV** (Bradski, G., 2007)**:** To load and handle the images.
- **Scikit-learn** (*Scikit-Learn*, 2011)**:** For its robust implementation of the K-Means clustering algorithm.
- **Matplotlib** (J. D. Hunter, 2007)**:** To visualize the final color palettes.
- **Numpy** (*NumPy Documentation — NumPy V2.3 Manual*, n.d.)**:** For efficient numerical operations.

It can be installed easily using pip:

```
pip install opencv-python
pip install scikit-learn
pip install matplotlib
pip install numpy
```

3. The Code: Extracting the Palette Step-by-Step

That's an intuitive Python script that ingests an image and outputs its dominant color palette along with the percentage of each color. The logic is straightforward: it will treat every pixel in the painting as a data point and use the K-Means algorithm to group these pixels into a small number of color clusters. The center of each cluster will be a dominant color in the final palette. So, it can be started importing those libraries:

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from collections import Counter
```

The core of this project is encapsulated in a single Python function, `extract_color_palette`. This function handles everything from loading the image to displaying the final color signature. Let's break down its internal workings block by block.

First, the function is defined to accept the path to an image and the desired number of colors for the palette.

```
def extract_color_palette(image_path, number_of_colors, show_chart=True):
    """
    Extracts and quantifies the dominant color palette from an image using
K-Means clustering.

    :param image_path: The path to the image file.
    :param number_of_colors: The number of dominant colors to extract.
    :param show_chart: If True, displays a chart of the color palette.
    :return: A dictionary of colors (R, G, B) and their percentage.
    """
```

3.1. Image Loading and Pre-processing

Before any analysis can occur, the image must be loaded into memory and prepared for the machine learning algorithm. This involves four key actions: loading, color correction, resizing, and reshaping.

```
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
resized_image = cv2.resize(image,
(100,100),interpolation=cv2.INTER_AREA)
pixels =resized_image.reshape(-1, 3)
```

First, the image is loaded from the specified `image_path` using OpenCV. A crucial conversion from BGR to RGB color space is performed, as OpenCV defaults to BGR, while most other image libraries (including Matplotlib for our final chart) use the standard RGB format.

To ensure efficient processing, the image is then resized to a standard 100x100 pixels. This step significantly speeds up the algorithm's execution time without losing the essential color information. Finally, the image's 2D structure is flattened into a one-dimensional list of pixels, which is the required input format for the Scikit-learn implementation of K-Means.

3.2. Applying the K-Means Algorithm

With the data prepared, the core of the analysis can be performed. The K-Means clustering algorithm is an unsupervised learning model whose goal is to partition data points into a predefined number of clusters. In our case, the data points are the pixels, and the clusters are the dominant colors.

```
kmeans = KMeans(n_clusters=number_of_colors, random_state=42,
n_init=10)
kmeans.fit(pixels)
```

An instance of the `KMeans` model is created with the desired number of clusters (`number_of_colors`). The model is then "fitted" to our pixel data. During this process, the algorithm identifies the centroids—the geometric center of each cluster—which represent the average color for that group. These centroids will become the colors of our final palette.

3.3. Quantifying The Results

After the model is trained, the next step is to interpret its output. This involves counting the number of pixels assigned to each color cluster and converting this count into a percentage, giving us the quantitative distribution of the color signature.

```
total_pixels = len(kmeans.labels_)
    counts = Counter(kmeans.labels_)

    sorted_counts = sorted(counts.items(), key=lambda x: x[1],
reverse=True)

    palette = {}
    dominant_colors_rgb = []
    percentages = []

    for cluster_index, count in sorted_counts:
        color_rgb =
tuple(kmeans.cluster_centers_[cluster_index].astype('uint8'))
        percentage = (count / total_pixels) * 100
        palette[color_rgb] = percentage
        dominant_colors_rgb.append(color_rgb)
        percentages.append(percentage)
```

The code first calculates the total number of pixels and counts how many belong to each cluster identified by the `kmeans.labels_`. It then iterates through these clusters, from the most populated to the least, extracting the RGB value of the centroid and calculating its percentage.

3.4. Visualizing the Color Signature

Finally, data is often most powerful when visualized. This last block of code uses Matplotlib to generate a clean and intuitive bar chart that displays each dominant color and its corresponding percentage in the painting.

```
if show_chart:
        plt.figure(figsize=(10, 5))
        bars = plt.bar(range(number_of_colors), percentages,
color=[tuple(c/255.0 for c in color) for color in
dominant_colors_rgb])

        plt.title(f'Color Signature of:
{image_path.split("/")[-1]}')
        plt.ylabel('Percentage (%)')
        plt.xticks([])
```

```
        for i, bar in enumerate(bars):
            plt.text(bar.get_x() + bar.get_width() / 2,
bar.get_height() + 1, f'{percentages[i]:.1f}%', ha='center',
va='bottom')

        plt.ylim(0, max(percentages) + 10)
        plt.show()

    return palette
```

This code generates a bar chart where the color of each bar corresponds to a dominant color from the palette and its height represents its prevalence in the image. The percentages are also printed on top of each bar for clarity, creating a complete and easy-to-read "Color Signature."

4. The Use Case: The Signatures of the Master

To demonstrate the power of this analytical approach, we will apply our Python script to a few famous paintings, creating a comparative analysis of their unique color signatures.

4.1. Vincent van Gogh's "The Starry Night"

First, the script is applied to one of the most famous and emotionally charged paintings in history. Van Gogh's work (*Vincent Van Gogh. The Starry Night. Saint Rémy, June 1889*, n.d.) is known for its intense energy, and "The Starry Night" is a prime example of his unique use of color to convey deep feeling.

Figure 1: "The Starry Night" (1889) by Vincent van Gogh (Image from MoMA Collections).

Running the `color_palette_analyzer.py` script on this masterpiece yields a revealing color signature.

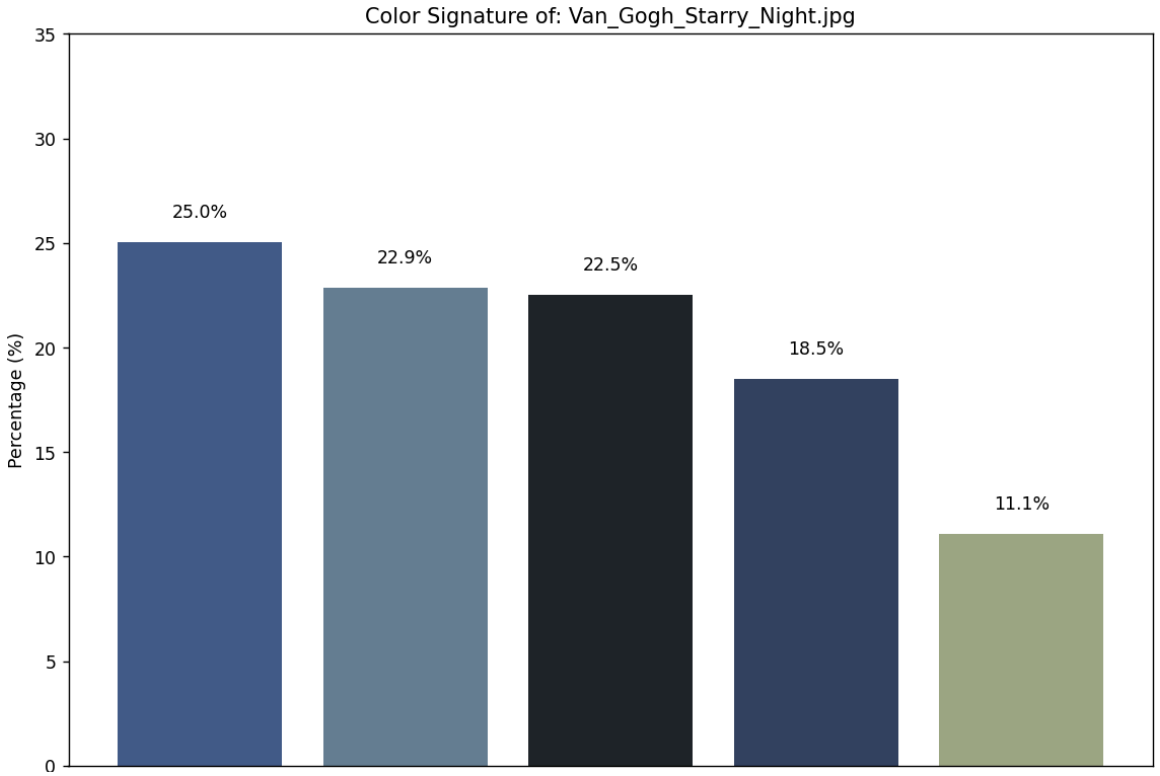

Figure 2: The Color Signature of "The Starry Night", showing the 5 most dominant colors and their percentage (Image by the author).

The bar chart immediately confirms what the eye suspects, but with mathematical certainty. The painting is overwhelmingly dominated by a triad of dark tones; nearly **89% of the palette is composed of deep blues and black**. This data quantifies the melancholic atmosphere and the immense, heavy weight of the night sky that Van Gogh depicted. In stark contrast, the light of the stars and the moon, represented by the muted yellow-green tone, constitutes a mere **11.1% of the palette**.

What this objective data suggests is the profound drama at the heart of Van Gogh's work. The light, though visually explosive and energetic, is in a constant struggle against a predominant darkness. The analysis shows how a simple machine learning model doesn't diminish the art; instead, it provides a new language to appreciate its emotional tension. We can now objectively measure the beautiful and desperate struggle between light and dark that makes this painting a timeless masterpiece.

4.2. Claude Monet's "Impression, Sunrise"

To provide a point of contrast, let's analyze a masterpiece that defined a whole new movement: Monet's "Impression, Sunrise." (*Impression, Soleil Levant (Rising Sun), 1872 - Claude Monet — Google Arts & Culture*, n.d.) While Van Gogh uses color to express intense emotion, Monet uses it to capture the fleeting qualities of light.

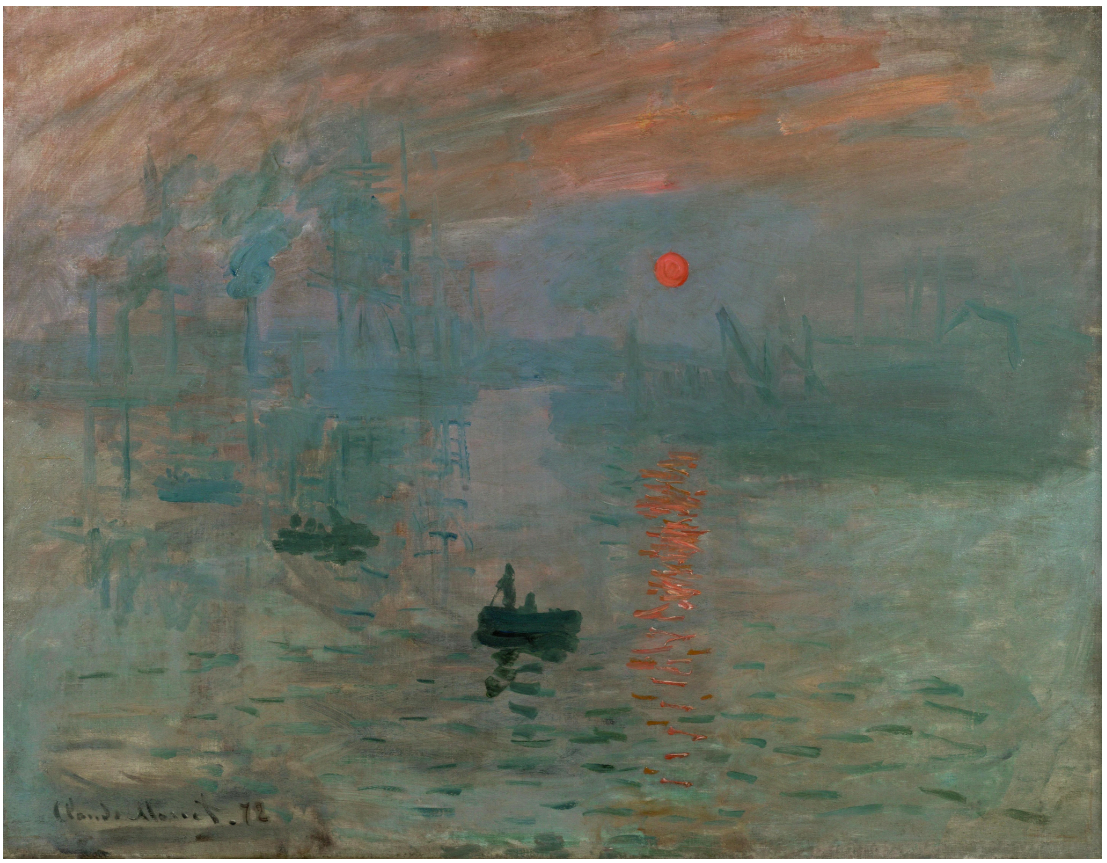

Figure 3. "Impression, Sunrise" (1872) by Claude Monet (Image from *Google Arts & Culture*).

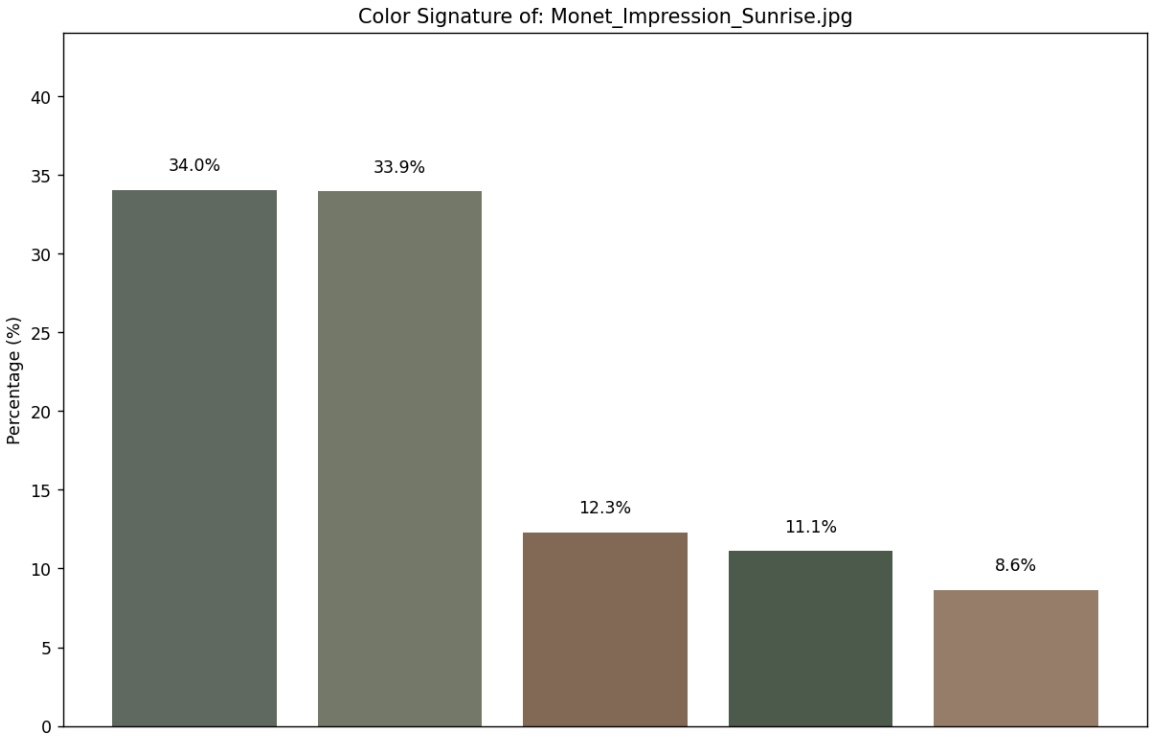

Figure 4. The Color Signature of "Impression, Sunrise" (Image by the author).

The resulting color signature is dramatically different. Where Van Gogh's palette was dominated by dark, saturated blues, Monet's is a study in subtlety and light. The signature is composed mainly of muted greys, soft blues, and pastel tones, which quantitatively represent the hazy, humid atmosphere of the port at dawn.

The most striking feature is the small but vital percentage of vibrant orange. This single data point is the famous sun, the focal point of the entire painting. Our analysis shows that its impact is not due to its dominance in quantity, but to its sharp contrast against the otherwise low-saturation palette. Here, our tool helps us objectively measure the "impression" that Monet so brilliantly crafted.

4.3. A Personal Study: "Path Under The Flamboyant"

Finally, this tool can be used as a powerful method for self-reflection. To conclude our analysis, I applied the script to one of my own paintings, a landscape dominated by the fiery red of a Flamboyant tree (Paloma Sette, 2009).

Figure 5: "Path Under The Flamboyant" (2009), Paloma Sette (Image by the author).

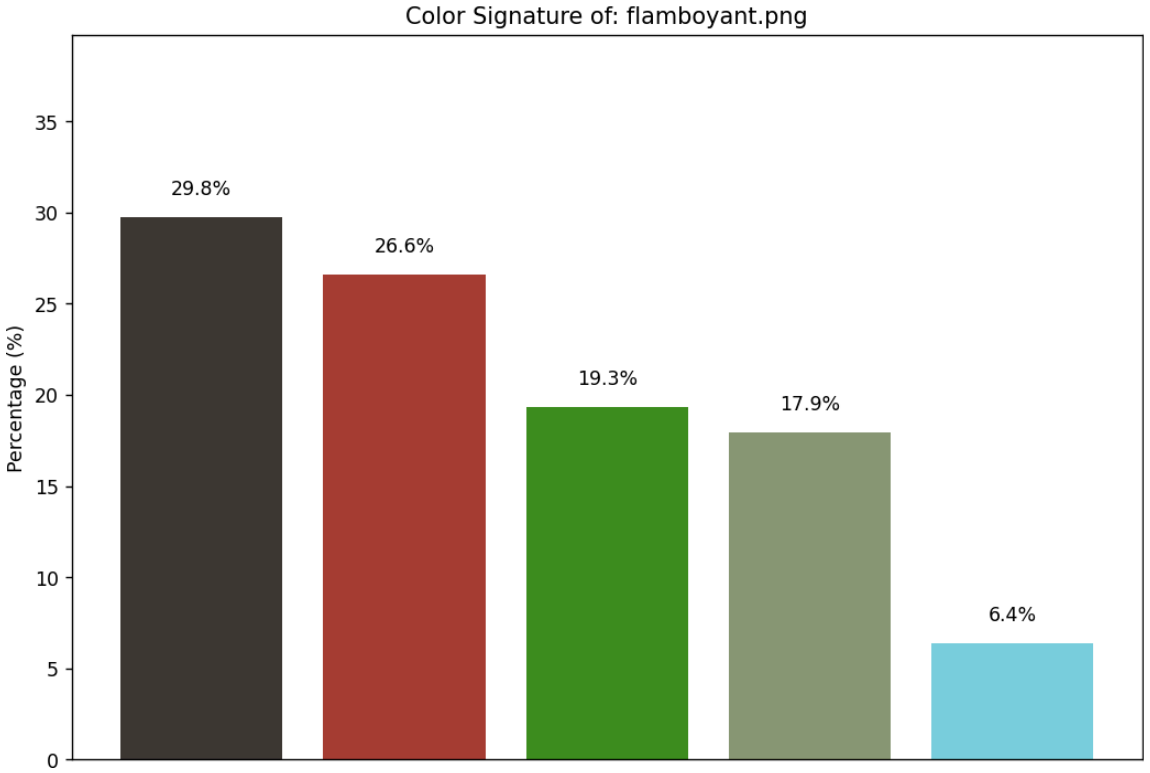

Figure 6: The Color Signature of the Path Under The Flamboyant painting (Image by the author).

The result was fascinating, and quite frankly, surprising. My own perception was that the painting was dominated by the vibrant reds and greens. However, the data reveals a different story. The most dominant color, at nearly **30%**, is a dark, earthy brown-red, originating from the deep shadows and tree trunks. The bright red of the flowers, the painting's focal point, comes in second at **26.6%**.

This discrepancy is the most valuable insight of this entire experiment. It highlights the difference between objective, computational analysis and subjective, human perception. The human brain is wired to prioritize high-contrast, high-saturation elements (the red flowers), giving them more emotional and perceptual "weight." The algorithm, free from this bias, simply counts the pixels, revealing the foundational role of the dark tones in creating the painting's structure and depth.

This proves the value of the tool not as a replacement for the artist's eye, but as a surprising mirror. It can reveal the unconscious choices made, quantifying the hidden architecture of our own work and uncovering the mathematical truth behind the magic of a painting.

Conclusion

The journey from a philosophical question to a practical Python script demonstrates the powerful synergy between art and data science. The use cases, from Van Gogh's emotional

intensity to a surprising self-analysis, revealed a key insight: a simple algorithm can offer a profoundly new perspective on art, not only of the masters but also of our own.

The color signature tool does not tell us if a painting is "good," nor does it explain away its mystery. Instead, it provides an objective dataset that enriches our subjective interpretation. The most fascinating discovery was not just quantifying the colors, but revealing our own perceptual biases. We learned that the colors our brain prioritizes for their emotional impact are not always the ones that are most mathematically present.

The most exciting future for Artificial Intelligence may not lie in its ability to create autonomously, but in its potential to augment our own capacity for understanding, appreciation, and introspection. By building these bridges between the technical and the artistic, we don't just analyze art; we enrich our dialogue with it, and with ourselves.

FAQs

1. What is K-Means clustering and why is it used for colors? K-Means is an unsupervised learning algorithm that groups data points into a specified number of clusters (k). It's an excellent choice for color quantification because it treats every pixel as a data point and effectively identifies the most common color groups by finding their average center, which becomes a dominant color in our palette.

2. Why did you resize the image before processing? Resizing the image to a smaller, standard size (e.g., 100x100 pixels) drastically speeds up the K-Means algorithm's computation time. A high-resolution image can have millions of pixels, which would be very slow to process. For extracting a general color palette, the overall color distribution is more important than fine detail, making resizing an effective optimization.

3. Can I use this code on my own photos? Absolutely! The script will work on any image file. It can be a fascinating way to discover the dominant color palette of your own photography, a brand logo, or even a frame from a movie.

4. How does the number of colors (the 'k' in K-Means) affect the result? The `number_of_colors` (k) is a crucial parameter. A small `k` (like 3 or 4) will give you a very broad, high-level summary of the color scheme. A larger `k` (like 8 or 10) will capture more subtle variations and secondary colors. The ideal number depends on the complexity of the image and the level of detail you wish to analyze.

5. What is the difference between this method and a "color picker" tool? A color picker tool manually samples the color of a single pixel you click on. This method, by contrast, analyzes *all* the pixels in the image simultaneously and calculates which color groups are the most statistically significant. It gives you a true representation of the overall color balance, not just an individual shade.

6. Where can I find high-quality images of artworks to analyze? Great sources for high-resolution, public-domain images of art include the Google Arts & Culture platform, the online collections of major museums like The Metropolitan Museum of Art (The Met), the Rijksmuseum, and the Art Institute of Chicago. Always check the copyright and usage rights for each image.

References

Bradski, G. (2007). *The OpenCV Library*. OpenCV - Open Computer Vision Library. Retrieved September 13, 2025, from http://opencv.org

Impression, Soleil Levant (Rising Sun), 1872 - Claude Monet — Google Arts & Culture. (n.d.). Google Arts & Culture. Retrieved September 13, 2025, from https://artsandculture.google.com/asset/impression-soleil-levant-rising-sun-1872-claude-monet/awGukZXBkfTvEQ?hl=en

J. D. Hunter. (2007). *Matplotlib: A 2D graphics environment*. Matplotlib — Visualization with Python. Retrieved September 13, 2025, from https://matplotlib.org

NumPy documentation — NumPy v2.3 Manual. (n.d.). NumPy. Retrieved September 13, 2025, from https://numpy.org/doc/stable

OpenCV. (2000). OpenCV - Open Computer Vision Library. Retrieved September 13, 2025, from http://opencv.org

Paloma Sette. (2009). *Path Under The Flamboyant (Caminho sob o Flamboyant)*. Brazil.

Scikit-Learn. (2011). scikit-learn: machine learning in Python — scikit-learn 1.7.2 documentation. Retrieved September 13, 2025, from http://scikit-learn.org

2.3. Clustering — scikit-learn 1.7.2 documentation. (n.d.). Scikit-learn. Retrieved September 13, 2025, from https://scikit-learn.org/stable/modules/clustering.html