Anna Makarewicz
CSC369: Intro to Distributed Computing
10 February 2025

Week 5 Analysis – Tell me something I don't know

After hearing some chatter in the class that there could be a Cal Poly logo on the r/place canvas, my goal was to identify the presence of Cal Poly-related elements on the r/place canvas by focusing on the color Cal Poly Mustang green (#154734 or (21, 71, 52)) and analyzing how closely other pixels on the canvas match it. I first identified the official Cal Poly colors and created a filter list to check for the presence of these colors in the dataset. These colors were based on the official Cal Poly color palette.

```
filter_list = ["#154734", "#BD8B13", "#3A913F", "#A4D65E", "#F2C75C", "#F8E08E",
"#5CB8B2", "#B5E3D8", "#ABCAE9", "#D5E4F4",
               "#CAC7A7", "#E4E3D3", "#B7CDC2", "#789F90", "#54585A", "#8E9089",
"#D0DF00", "#FF6A39"]
df_filtered = df.filter(col("pixel_color").isin(filter_list))
```

Since the Cal Poly Mustang green was not found directly in the dataset, I implemented a method to identify pixels that are close to the official Cal Poly green color using a color distance approach. Instead of relying on a simple Euclidean distance, I added a bias towards the green component to eliminate colors with low green values or shades of gray.

To find the closest matching green pixel color, I converted the hex values to RGB tuples.

```
def hex_to_rgb(hex_color):
    hex_color = hex_color.lstrip('#')  # Remove '#' if present
    return tuple(int(hex_color[i:i+2], 16) for i in (0, 2, 4))
```

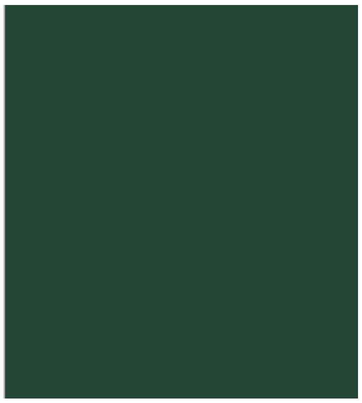
Then, I implemented a function that calculates the Euclidean distance between two colors represented in RGB. I also included an additional check to ensure the green component is significantly higher than both the red and blue components. Only pixels with enough green were considered, and shades of non-green colors were filtered out.

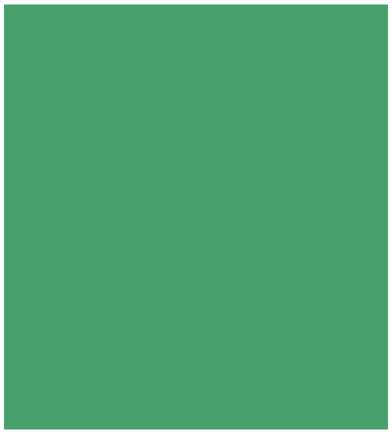
```
def refined_color_distance(hex_color, target_hex, min_green_value=50,
green_threshold=6):
    r1, g1, b1 = hex_to_rgb(hex_color)
    r2, g2, b2 = hex_to_rgb(target_hex)

    # Ensure that green is significantly higher than red and blue
    if g1 < min_green_value or (g1 <= r1 + green_threshold) or (g1 <= b1 +
green_threshold):
        return float('inf')  # Assign a very high distance to non-green pixels
    # Calculate Euclidean distance for color matching
    return math.sqrt((r1 - r2)**2 + (g1 - g2)**2 + (b1 - b2)**2)
```

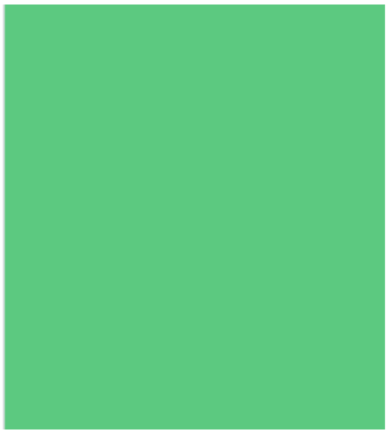
This function was converted into a PySpark UDF and applied to the dataset to compute the color distance between each pixel and the target Cal Poly Mustang green color. After calculating the color distances, the dataset was sorted by distance to easily identify the closest colors.

```
df_with_refined_distance = df.withColumn("color_distance",
refined_color_distance_udf("pixel_color"))
df_sorted = df_with_refined_distance.orderBy("color_distance")
```

I found two greens that were relatively close to Mustang Green. The closest green to Cal Poly Mustang green was #00A368 (or (0, 163, 104)), and the calculation had a distance of 107.74507. The next closest green to Cal Poly Mustang green was #00CC78 (or (0, 204, 120)), and the calculation had a distance of 150.84428.


Cal Poly Mustang Green
#154734


#00A368


#00CC78

PySpark and Polars Comparison

Both PySpark and Polars are powerful libraries for data processing and analysis. The main difference I found is that Polars is often much faster and easier to use than PySpark.