Do colors convey political information?

```
# import modules
    #* from environment
import os # for path management
import sys # for path management
import numpy as np # for array management
import polars as pl # for dataframe management
import cv2 # for color detection
from matplotlib import pyplot as plt # for graphics
from matplotlib import image as mpimg # for displaying images
from IPython.display import display, Markdown # for displaying inline code
    #* user-defined
sys.path.append("../../code/")
from fun import preProcess
```

The systematic use of colors in campaign branding

```
import os

os.system('../../code/03_capd_downloading_images.py')
os.system('../../code/04_mit_election_lab_merge.py')
os.system('../../code/05_color_detection.py')
```

- Descriptive analysis of the use of color in yard signs
- Consider using district level fixed effects in a regression to show District PID $\rightarrow$ Color selection

To examine whether the use of colors on yard signs vary in systematic ways, I collect images from the 2018, 2020, and 2022 Congressional elections for the House of Representatives across the United States. These yard signs are pulled together on one website by the Center for American Politics and Design[1]. From this website, I am able to extract over 1,100 images for these three elections. I then combine this information with district-level data provided by the MIT election lab on election returns for candidates in these House elections [2].

With these data, I detect the percentage of the "Republican Red" and "Democratic Blue" on the yard signs and examine whether the 5-year smooth moving average of Democratic candidate vote share in that given district correlate. The purpose of this analysis is to examine

[1]See: https://www.politicsanddesign.com/

[2]See: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/IG0UN2

the hypothesis that campaigns respond to the preferences of partisan voters and adjust their branding as a result. In this case, the branding being the color on the yard sign.

To provide an example of how the color detection works, I collected the GOP logo used on their official Twitter account during the 2022 midterm election cycle. I load this image and convert it to a three-dimensional array that contains information about the GBR (reversed RBG) values for the pixels in that image. I then resize the images to be a standardized 224 x 224 pixels. The computer is trained to detect a range of GBR values that encompass the official "Republican Red"[3]. For the broader exercise, I do it for the color white[4] and "Democratic blue"[5]. Once this range of values is specified, the computer detects the pixels that do not contain values within this pre-specified range and converts those values to represent the color black. Figure 1 presents this process.

```
# Define colors to detect
    #* White
        #** Not defined. Default for colorDetector()
    #* Red
republican_red = [232, 27, 35] # target color
red_lower = [93, 9, 12] # lower end of spectrum for red
red_higher = [237, 69, 75] # higher end of spectrum for red

# Load gop image to read
img = cv2.imread("../../data/chapter_1/gop_2022.png")

# Detect colors
    # create boundaries
boundaries = [([red_lower[2], red_lower[1], red_lower[0]],[red_higher[2], red_higher[1], r
    #print("boundaries correct")
    #print(boundaries)
    # get information about original image
    #width = img.shape[1] # get width of img
    #height = img.shape[0] # get get height of img
    # Transform image
img_transformed = preProcess(img)
    #print("img correct")
    # get information about transformed image
    #width_t = img_transformed.shape[1] # get width of img_transformed
    #height_t = img_transformed.shape[0] # get height of img_transformed
    # calculate scale
```

[3]lower values: (93, 9, 12), higher values: (236, 69, 75)
[4]upper and lower values: (255, 255, 255)
[5]lower values: (0, 18, 26), higher values: (102, 212, 255)

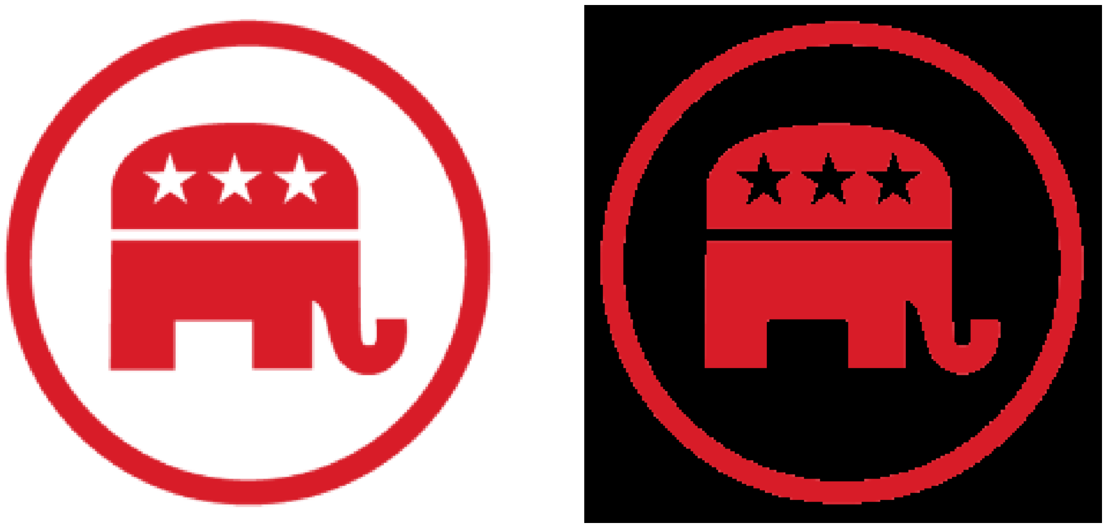
```
scale = ((img_transformed.shape[0]/img.shape[0]) + (img_transformed.shape[1]/img.shape[1])
    #print("scale correct")
for (lower, upper) in boundaries:
    # adjust red_higher and red_lower to specific type
    lower = np.array(lower, dtype = np.uint8)
    upper = np.array(upper, dtype = np.uint8)
    # take all values within color range and make it white; everything else black
    mask = cv2.inRange(img_transformed, lower, upper)
    # take all white values and put it on original image. keep black pixels black
    result = cv2.bitwise_and(img_transformed, img_transformed, mask = mask)
    # get ratio of non-black pixels
    ratio = cv2.countNonZero(mask)/(img_transformed.size/(1/scale))
    # calculate percentage of non-black pixels
    percent = (ratio * 100)/scale
cv2.imwrite("../../data/gop_2022_transformed.png", img_transformed) # save transformed ima

cv2.imwrite("../../data/gop_2022_detected.png", result) # save masked image

img_transformed = mpimg.imread("../../data/gop_2022_transformed.png") # load transformed i
plt.imshow(img_transformed) # show img_transformed
plt.axis("off") # remove axes
plt.show() # show plot

img_masked = mpimg.imread("../../data/gop_2022_detected.png") # load masked image
plt.imshow(img_masked) # show img_masked
plt.axis("off") # remove axes
plt.show() # show plot
```

I then extract the values in the array that are non-black and calculate the percentage of non-black pixels (as depicted in Equation 1).

$$\text{Color\%} = \frac{\text{Non-black}}{\text{Transformed}} \times \frac{\text{Original}_{\text{Height}} + \text{Original}_{\text{Width}}}{2\text{Transformed}_{\text{Height}} + 2\text{Transformed}_{\text{Width}}} \tag{1}$$

```
display(Markdown(""" For the example in @fig-color-detection-example, about {percent} of t
```

For the example in Figure 1, about 32.26 of the image is red.



(a) Resized original image (b) Masked

Figure 1: Detecting colors in the GOP logo