

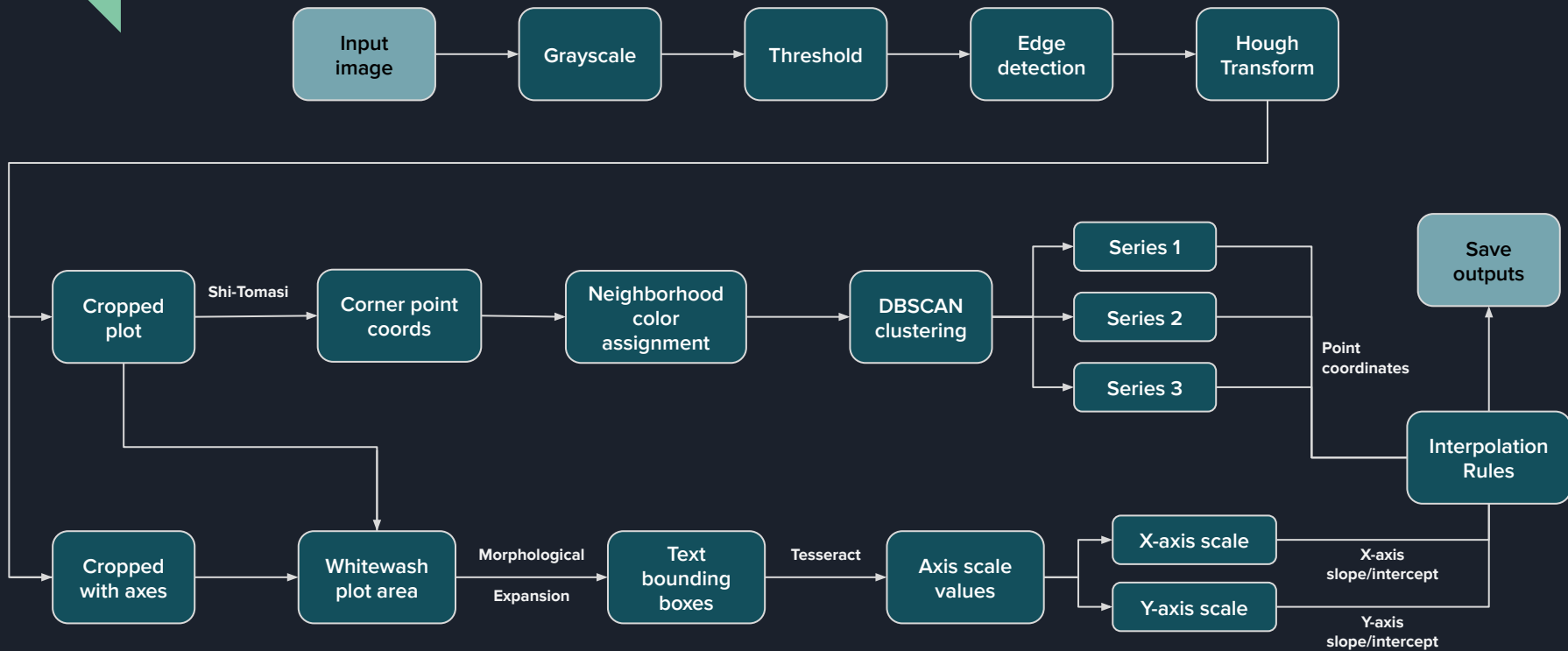


Astrazeneca AI Challenge

Team Gladiators, IIT Madras

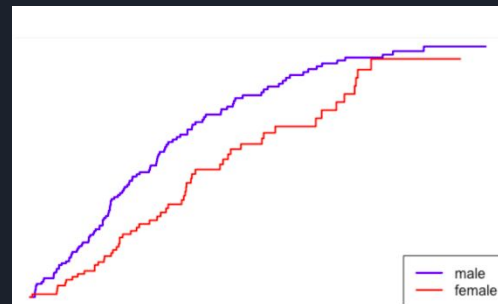
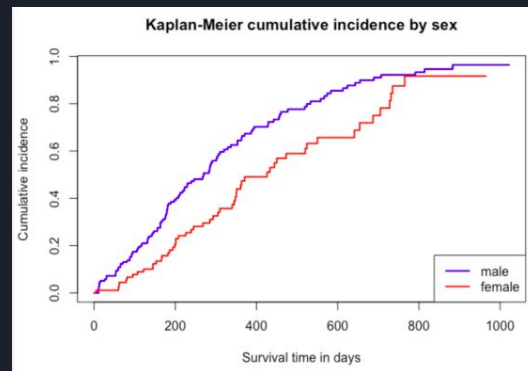
Nishant Prabhu, Vishwajit Hegde, Ojas Jain

Process Outline



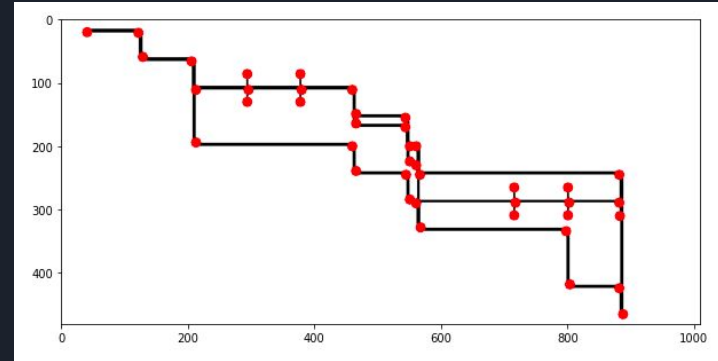
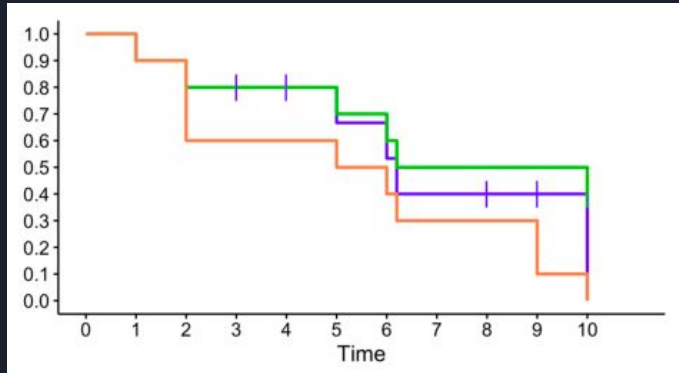
Initial processing

- Aim is to extract the plot region from the rest of the image
- Grayscale, thresholding and canny edge detection highlight plot axes clearly as continuous lines
- **Hough line transform** detects these lines and returns the coordinates of its end points
- Using these coordinates, plot region is cropped out of the image



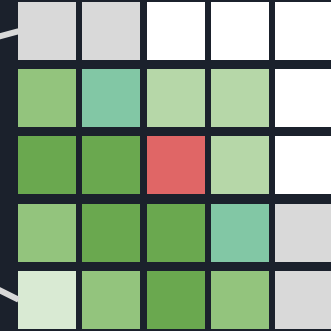
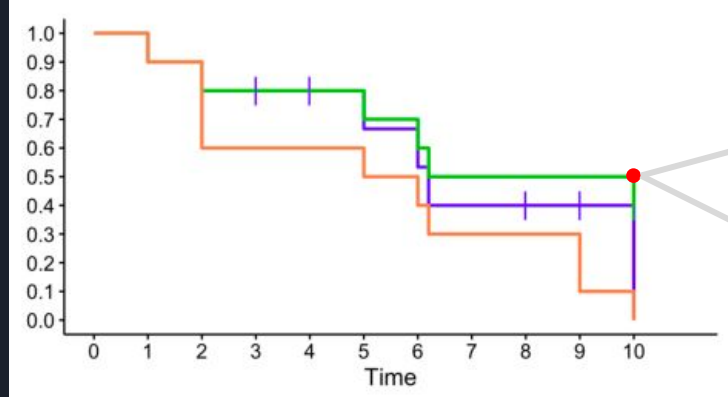
Series extraction pipeline

The cropped out image from the previous step is used here. First, we detect all the corner points using **Shi-Tomasi corner detection algorithm**. These are our candidate data points.



Series extraction pipeline


At each extracted corner point, we search a space of 5x5 pixels around the corner point for the most occurring color. This helps us assign each corner point to a color, and subsequently a series.





Series extraction pipeline

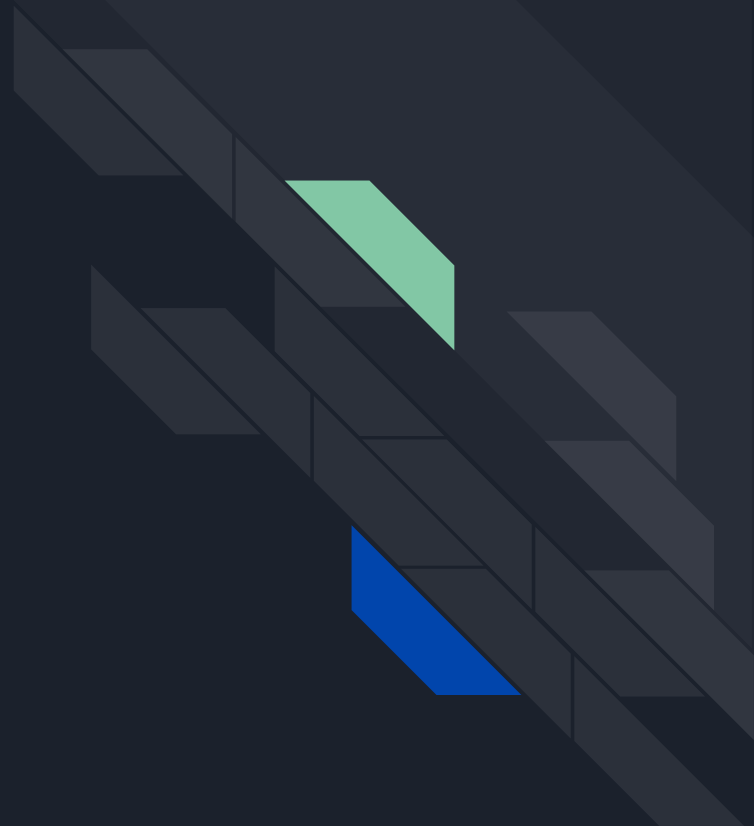
Now the colors assigned to each pixel are treated as data and clustering is performed on it using **DBSCAN**. This is a density based clustering algorithm, which means it can detect number of clusters on its own.

Colors_of_all_points = [Cluster_labels = [
[r1, g1, b1],		0,
[r2, g2, b2],		1,
[r3, g3, b3],		2,
....	
]]

At this point...

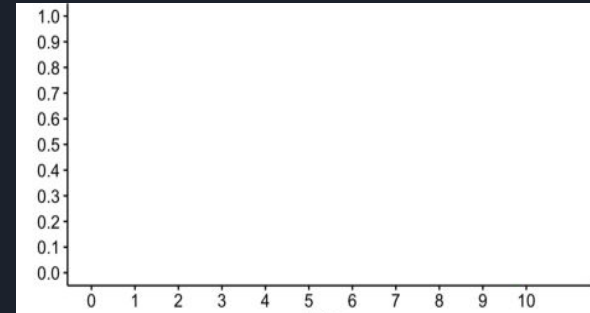
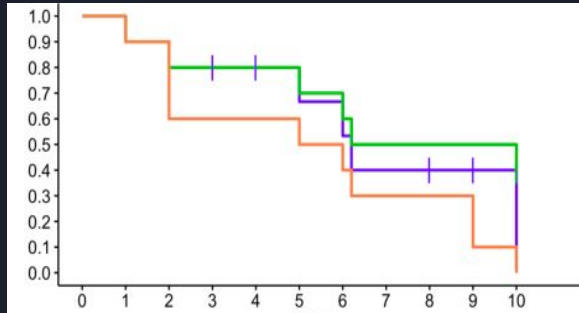
We have the coordinates of all our data points (of each series) separated by cluster labels.

Let's (try to) bring it down to actual scale.



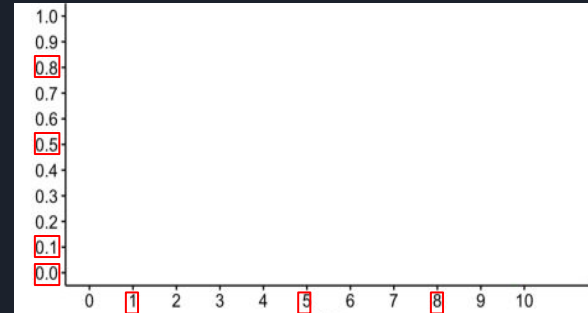
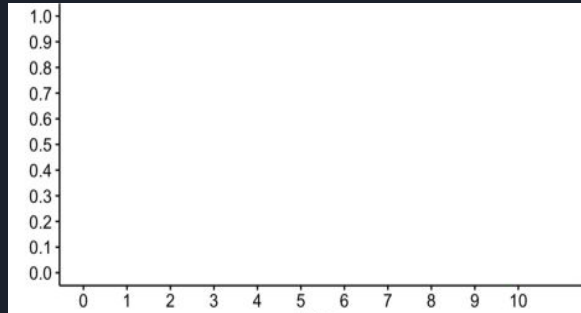
OCR & Interpolation pipeline

From the hough transforms output, we use the cropped plot and the cropped plot with axes included. In the second image, the region containing the cropped image is whitewashed. So now we only have the axes.



OCR & Interpolation pipeline

Now we perform a series of morphological transformations: elliptic convolution, gradient extraction (from convolution results), rectangular convolution, contour merging and bounding box generation. This gives us the bounding boxes around some of the text objects in the image.



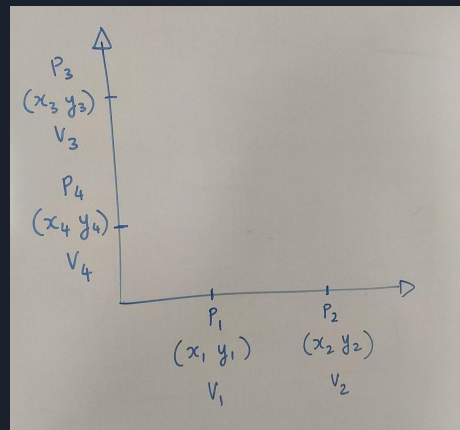
OCR & Interpolation pipeline

Each of the detected boxes are cropped out and converted to PIL images. These images are passed into **Tesseract OCR** module which returns the text contained by them.

Note. Since the size of the crops are small, OCR doesn't function accurately many number of times.

$$v_x = v_1 + \frac{v_2 - v_1}{x_2 - x_1} \cdot (x - x_1)$$

$$v_y = v_4 + \frac{v_3 - v_4}{y_3 - y_4} \cdot (y - y_4)$$





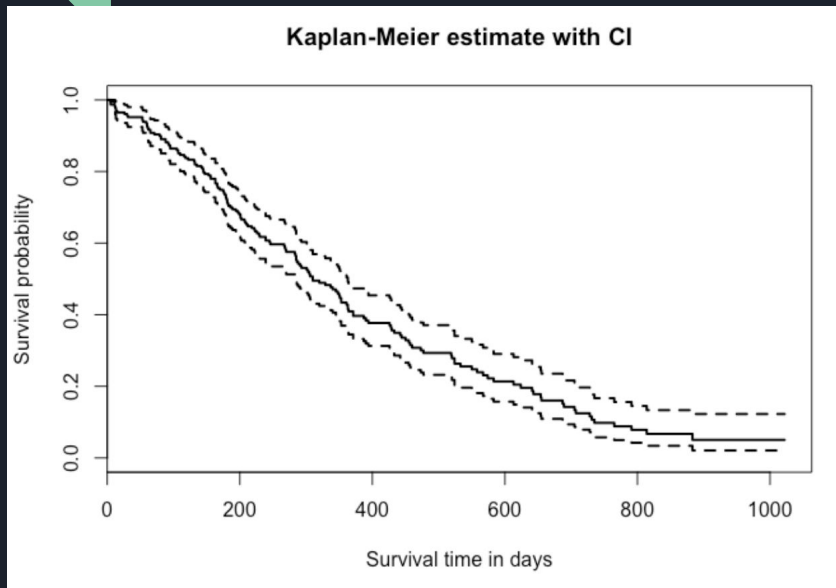
Finally ...

Using the coordinates of the points and the interpolation rules just defined, we can generate the actual points that made up the data.

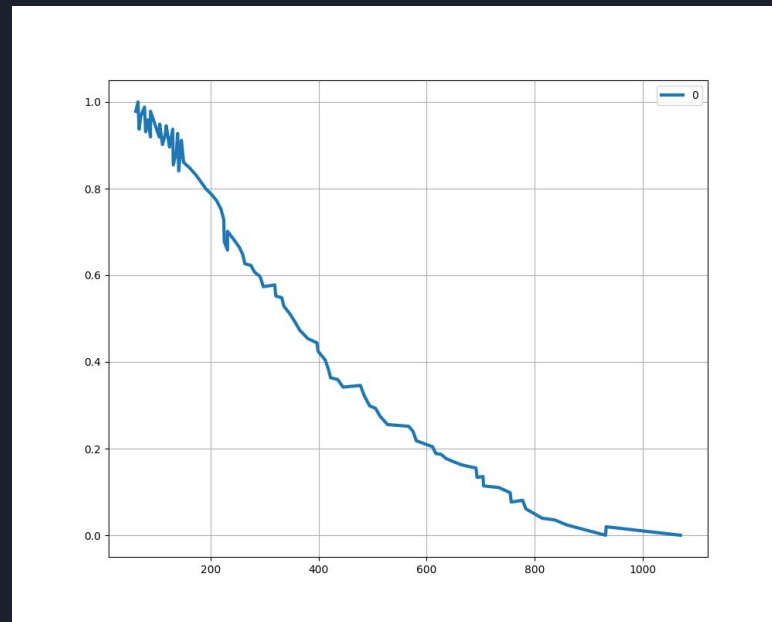
This is stored as CSV files in an auto-generated output folder in the project directory. The user also has options to generate the plots using extracted data for comparison.

Here are some results.

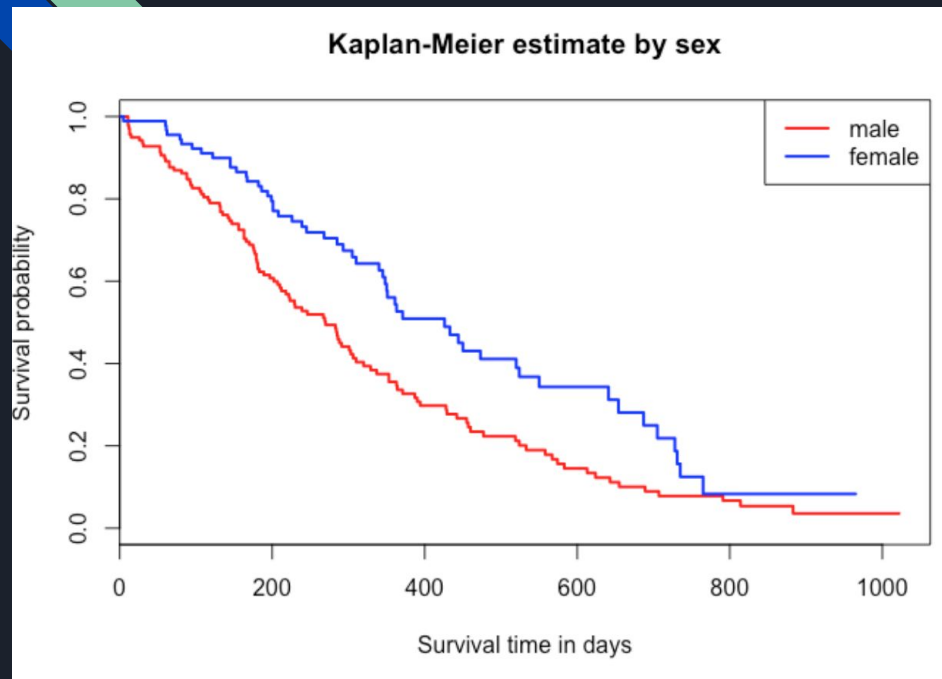




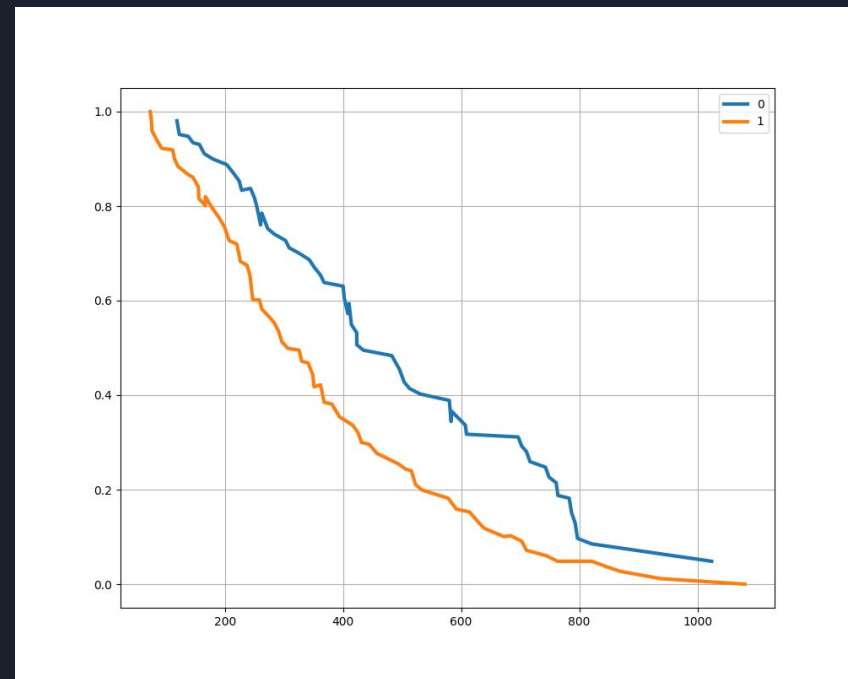
Real image



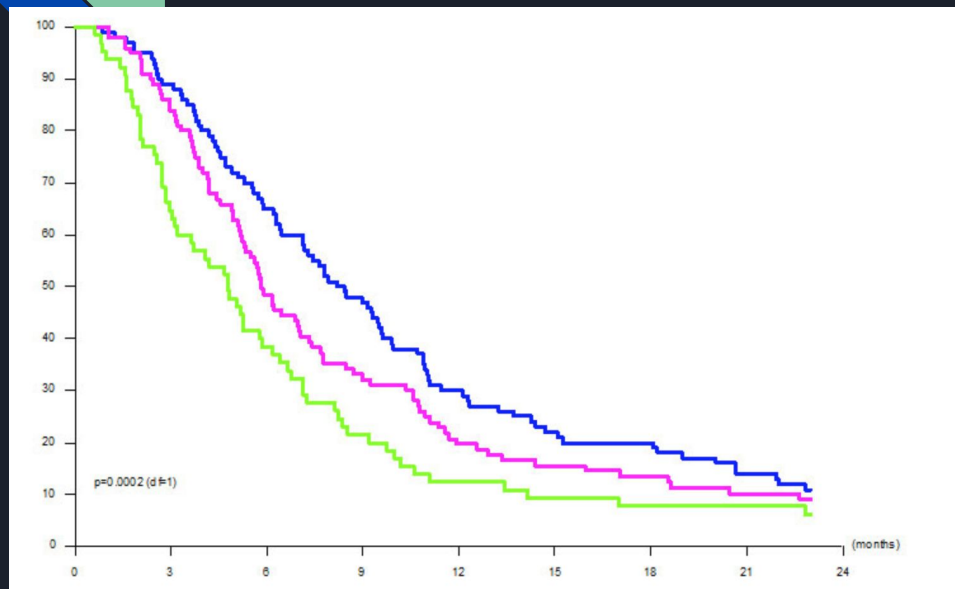
From extracted data



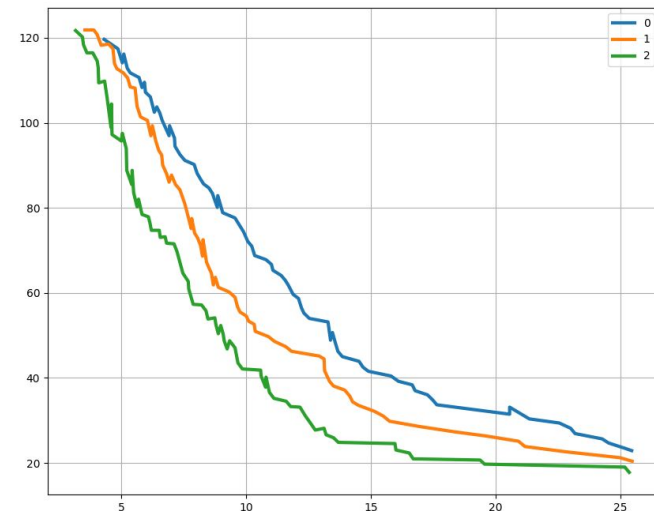
Real image



From extracted data



Real image



From extracted data
Axes scales are a little off in this one



Plus points of this algorithm

- Works out of the box. Doesn't need any training or preprocessing unlike deep learning algorithms.
- No extreme hardware requirements, can run on any decent machine.
- Fast inference, takes about half a minute to process the 20 input images.



Current known issues

- Inputs where individual plots intersect in several places are difficult to decode for this algorithm
- Tesseract OCR doesn't work very well on small sized images (crops of the number bounding boxes). Thus in many cases our axis scales are not as good as expected.
 - The algo scales that axis to lie between 0 and 1 in such cases.
- Bounding boxes are sometimes not detected for rotated text (such as numbers on the y-axis in some images)



Thanks a lot!