ASTRAZENECA AI CHALLENGE

SHAASTRA 2021

TEAM: AI PENGUINS

OMKAR NATH

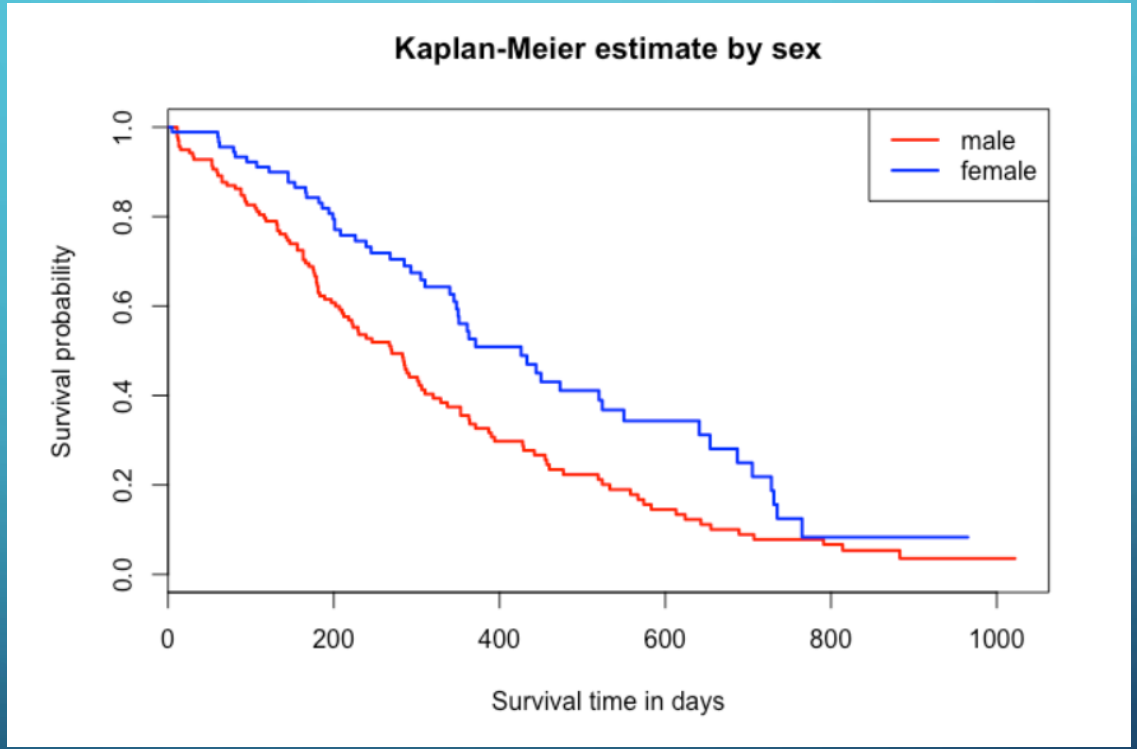
SHREYA NEMA

UMA T V

PROBLEM STATEMENT

- Build a program that is able to automatically reverse engineer a Kaplan Meier Chart
- Kaplan Meier Chart is the visual representation of depreciating or appreciate probability of an event with respect to time interval.
- The input format is an image of the chart
- Expected output is a csv/excel file consisting of the x-y data points on the chart.

EXAMPLE OF A KAPLAN MEIER CHART



APPROACH

- Our approach is to use a combination of image analysis techniques, and machine learning techniques, each with its specific applications.
- The broad approach is to identify key areas of the graph (axes, characters, data points) using image analysis
- This is followed by using machine learning for specific applications such as interpreting the textual data.

INTUITION FOR APPROACH

- Any image can be represented as a grid of pixels. Each pixel has an associated location with it in a 2-dimensional space the image
- The graph itself is meant to represent a 2-dimenstional space, as per the coordinate values of the two axes, x and y
- Use the positions of the curves in the pixel space to convert to coordinate space by creating a transformation between the two spaces.

APPROACH

1. Identify the unique colour groups used in the graph
2. Locate the x and y axes in the image, and their intersection.
3. Locate along the axes marking corresponding to values
4. Locate the regions of text corresponding to marked values.
5. Isolate each individual value of text along the axes
6. Use Machine Learning to read the text and obtain the values

APPROACH

7. Calculate the difference between the above markings
8. Define a transformation function between the two spaces.
9. Identify all possible curves along the graph
10. Track each curve to get all the data points corresponding to it
11. Apply the transforms to the curves to get the coordinates in x-y space
12. Save the data to a csv file for each curve.

KEY ASSUMPTIONS OF ALGORITHM

There are some key assumptions without which the algorithm won't work.

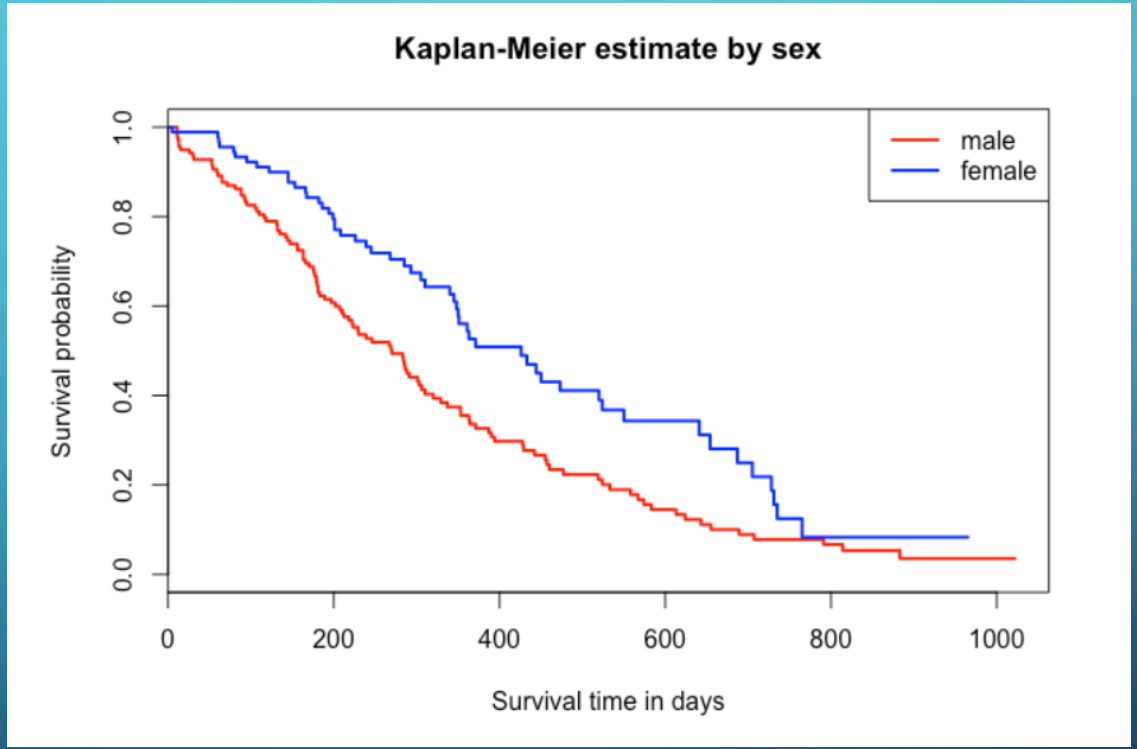
- The given image has x axis and y axis aligned with the borders of the image (so the image is not rotated basically)
- The graph occupies the majority of the image.
- There are markings along each of the axes for some of the points on the axis
- There are textual values corresponding to the markings that highlight the value of that position

SOME ASSUMPTIONS OF ALGORITHM

There are certain assumptions made in the current version of the algorithm. This can be resolved however by improving upon the algorithm

- The background is white, and the axes are black
- The x axis text is horizontal, while y axis text can be horizontal, or vertical
- Each curve has a unique and different colour.
- The markings are connected to the axes
- There is a slight gap between the axes markings and their textual values

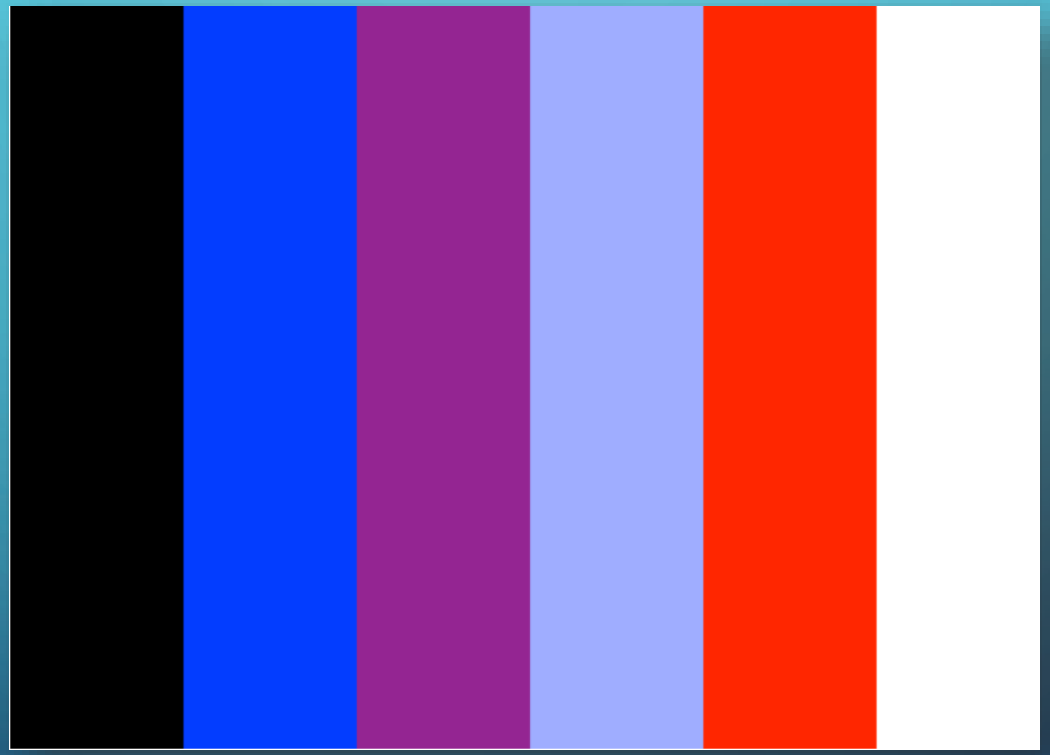
CONSIDER THIS EXAMPLE



COLOURS IN THE GRAPH

- Each colour doesn't have a unique pixel value.
- We want to identify the key colour "groups" that occur in the graph, around which most of the values cluster.
- Then based on which cluster the pixel is closer to, it can be classified as that colour.
- Groups are identified by taking maximas along a frequency distribution of the image

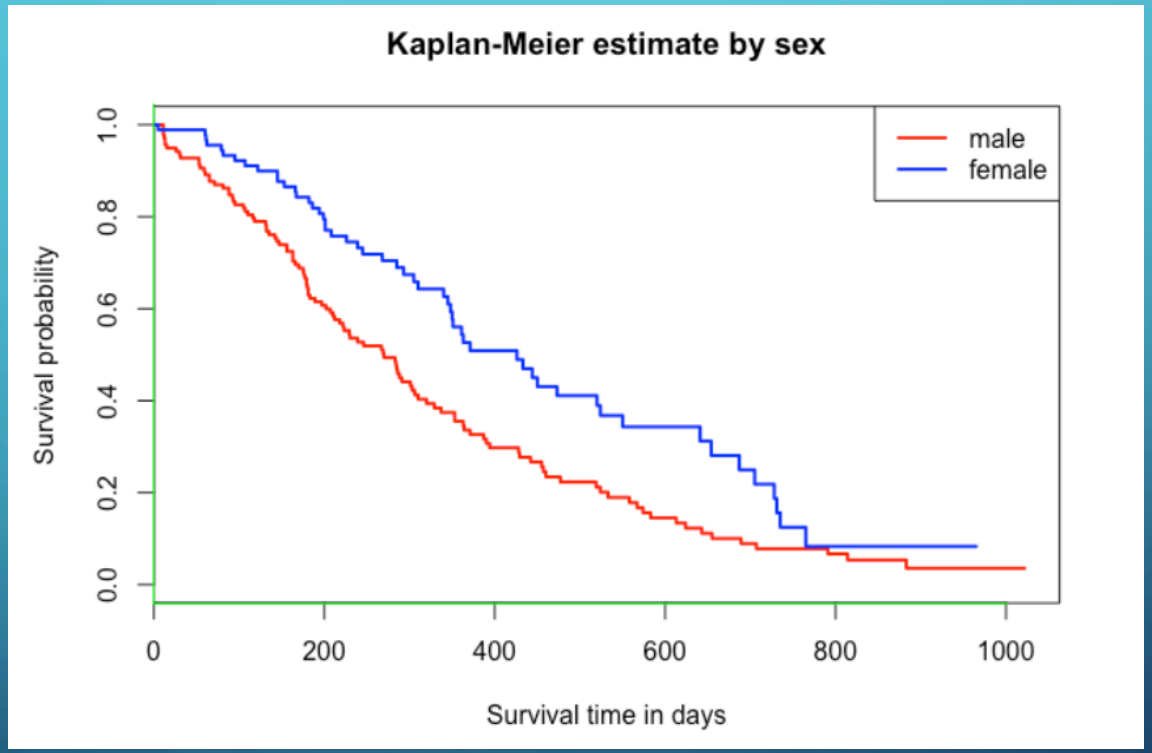
COLOURS IN THE GRAPH



LOCATING THE AXES

- This is done by finding a line along the x direction, which has a continuous strip of black pixels, of a minimum certain length based on the image dimensions.
- Similarly, for the y axis. The pixel coordinates of these axes are then noted.
- To ensure that we don't capture any borders outside of the graph, the search starts from the centre of the graph.

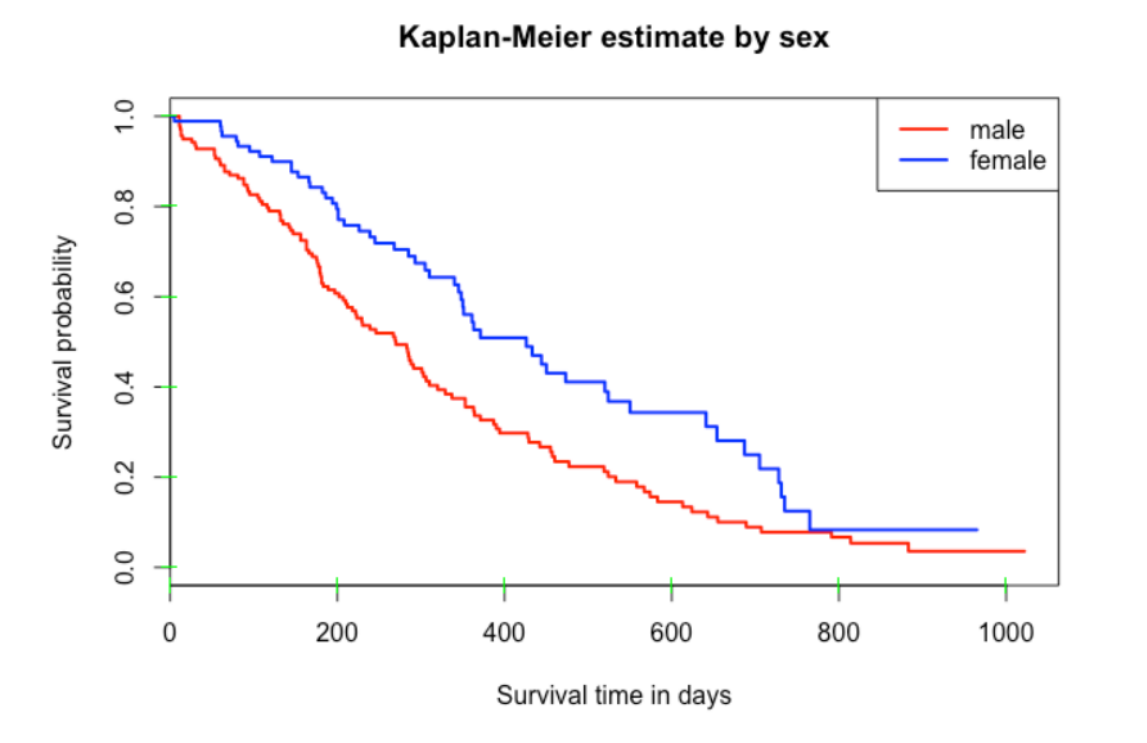
LOCATING THE AXES



FINDING AXES MARKINGS

- We start moving outwards from the axes until there are no more markings present (i.e., an entire row of white pixel space, before the textual values).
- Then we back trace a few pixels.
- Finally, the place where there are black pixels are noted and saved as the markings.

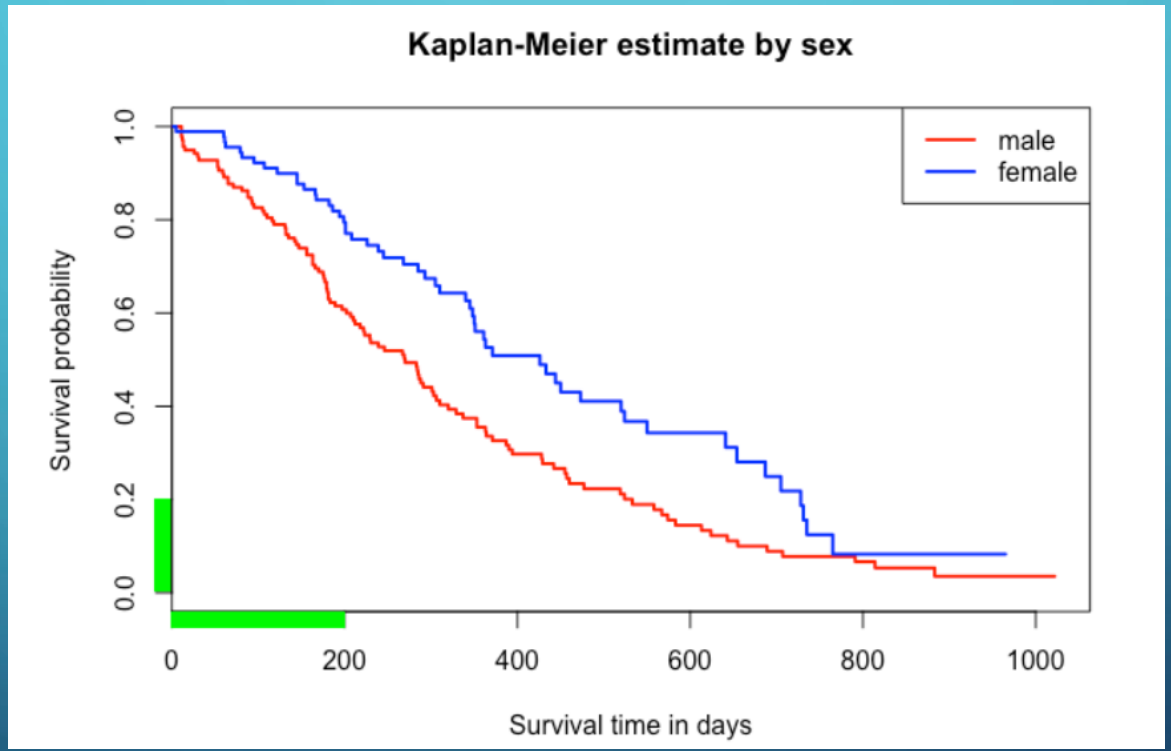
FINDING AXES MARKINGS



PIXEL SPACE DISTANCE FOR SCALE

- The distance between two divisions is taken by using the markings positions as earlier.
- This is taken by averaging over the entire axes.

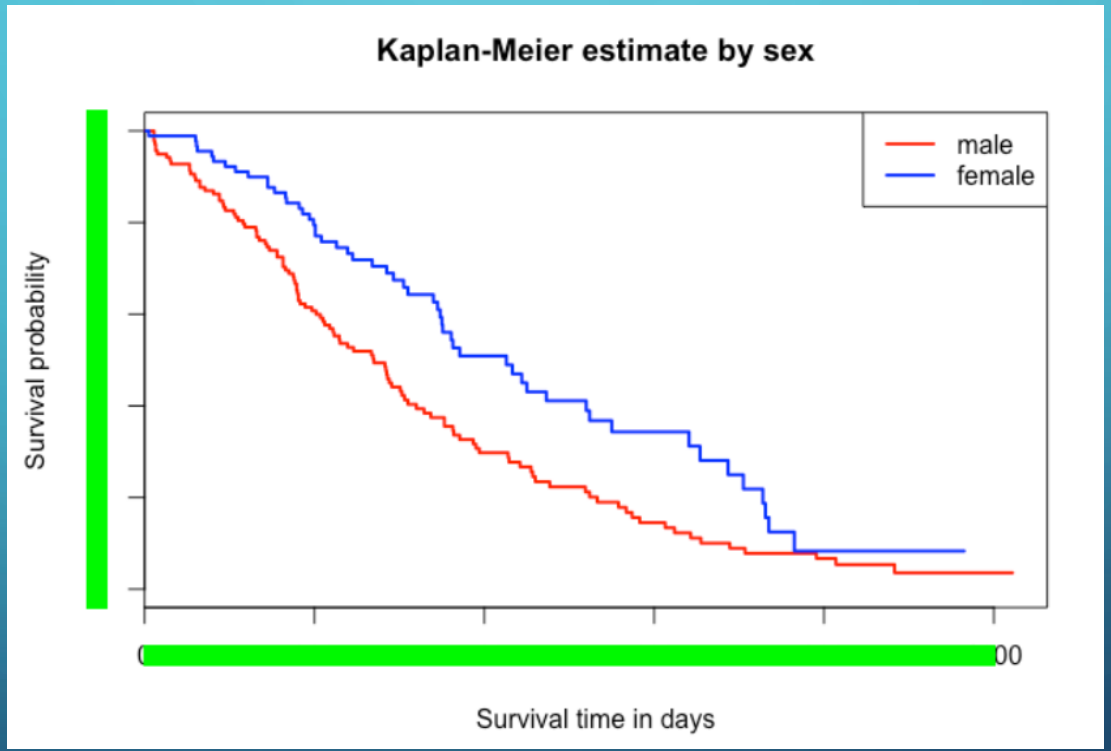
PIXEL SPACE DISTANCE FOR SCALE



LOCATING THE TEXT OF VALUES

- For this, it is assumed that just after the markings, there is white space located above and below the text.
- This white space is used as the guides for locating the top and bottom limits for the text.

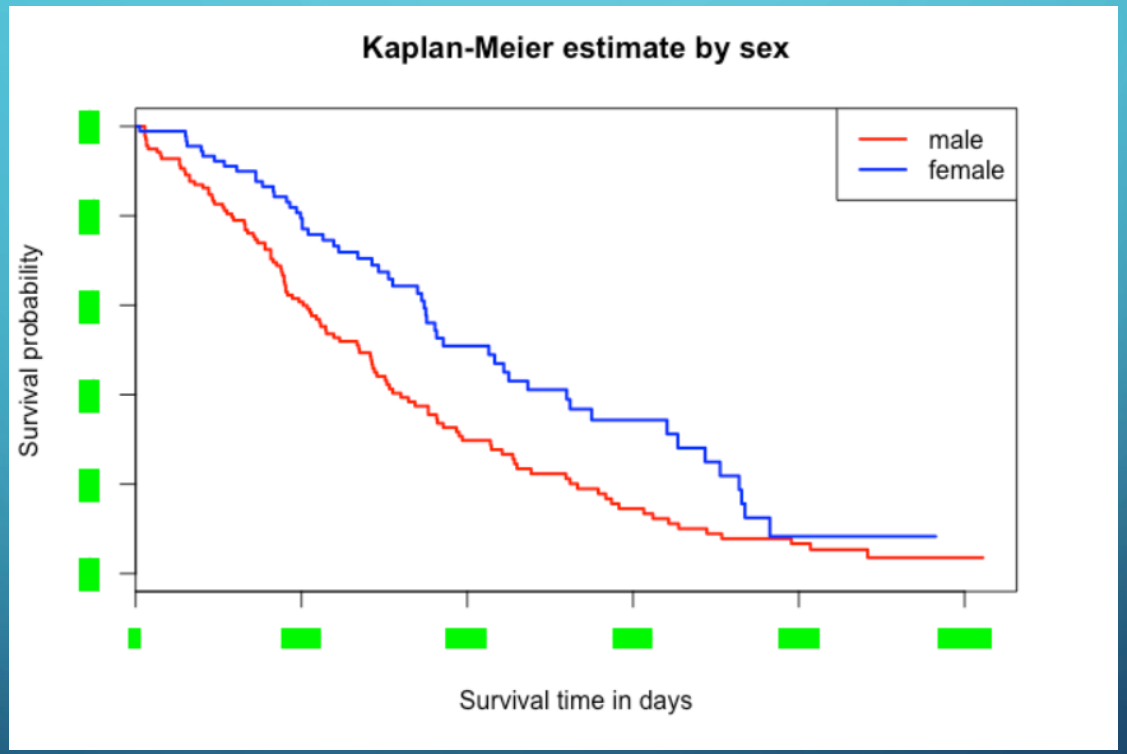
LOCATING THE TEXT OF VALUES



LOCATING THE TEXT OF VALUES

- Next is the left and right limits.
- This again can be found by simply seeing where the black pixel text ends.
- This can even be taken a step further to isolate each unique value, as per the white space located in between.

LOCATING THE TEXT OF VALUES



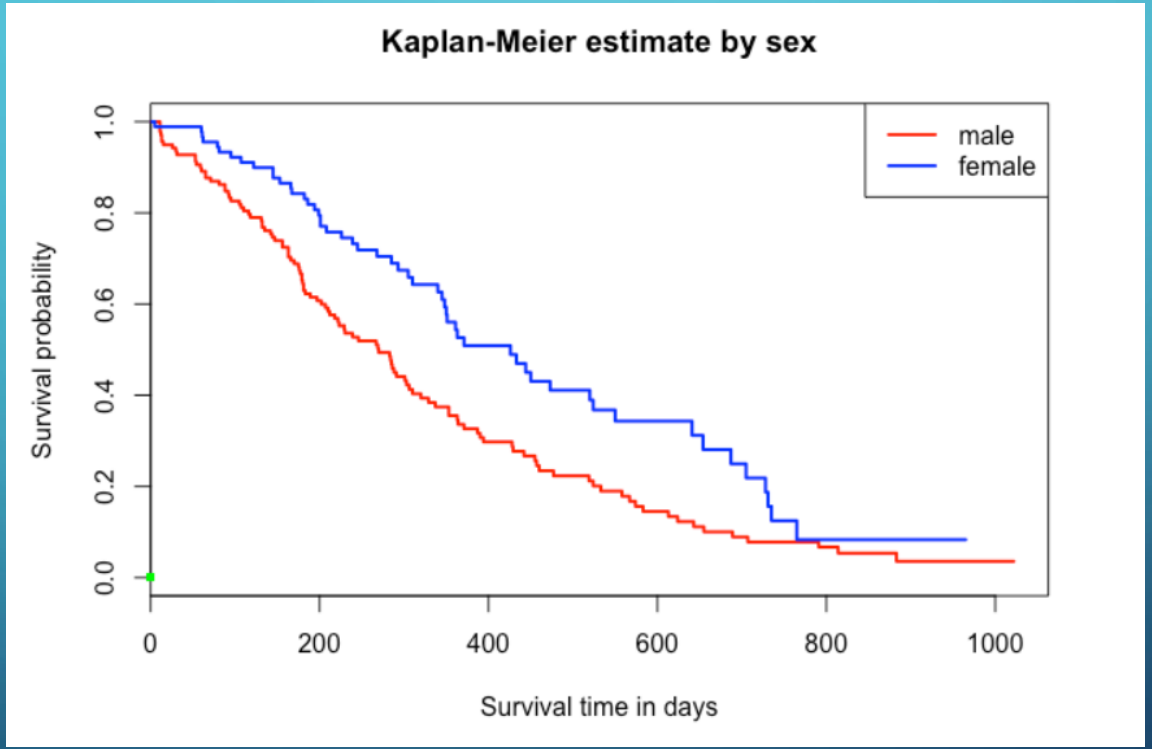
OPTICAL CHARACTER RECOGNITION

- The method we have chosen here is to use pytesseract package.
- The numerical values are obtained from pytesseract. This is used to estimate the scale, as well as locate the origin
- For y axis, the text may need to be rotated
- Pytesseract may not be the best package. There are better methods.

TRANSFORMING THE SPACES

- Having all the required information, a transformation function from pixel space to coordinate space can be defined
- Vice verse is also true
- Origin can also be found

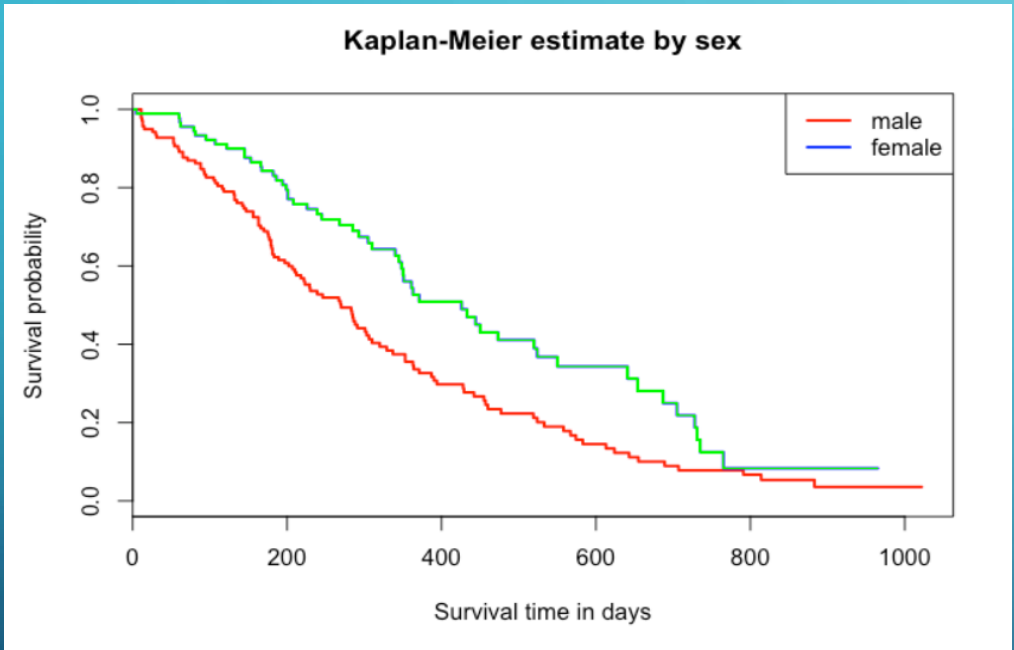
ACTUAL ORIGIN

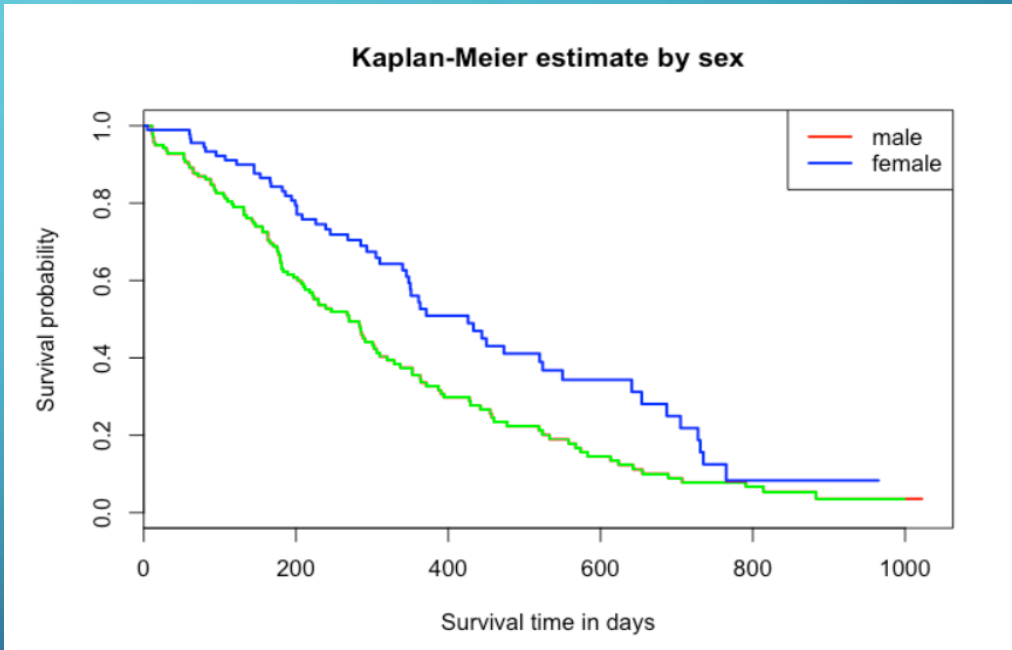


LOCATING THE POINTS ON THE CURVE

- For every possible colour value that exist within the appropriate region of the graph, the curve is tracked to ensure continuity, starting from left to right.
- Conditions on number of data points and nature of the curve are applied to ensure that what we have is a curve.
- Jumps are also allowed for the overlap of curves. This means that even dashed curves can be tracked.

LOCATING CURVES

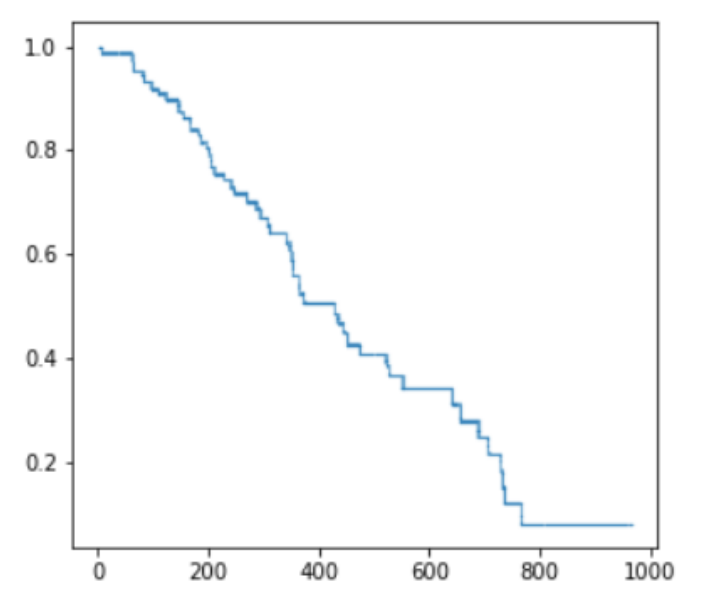


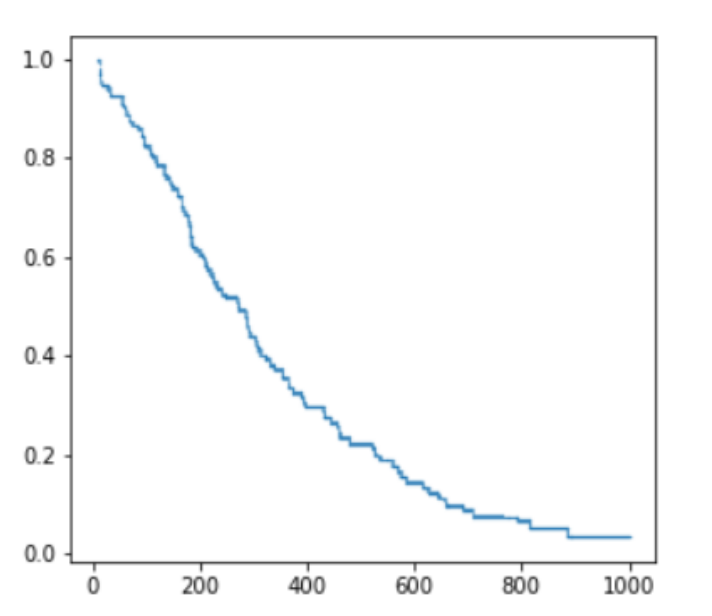


CONVERTING TO ACTUAL COORDINATE SPACE

- This can be simply done by using the transformation functions defined earlier. With this, we have our data points in x-y space. This is then saved to a csv file.
- As each pixel is a data point, each x coordinate has multiple y coordinates across different data points.
- There are also a lot of data points
- Further manipulations can be done as required

THE OBTAINED DATA POINTS





ADVANTAGES OF THIS APPROACH

- Algorithm can generalize well to different types of graphs
- It is able to handle multiple curves, as well as multiple types of curves, such as dashed curves
- There is no need to train the algorithm,
- The algorithm is reliable and controllable,
- It is highly efficient
- The basis of the algorithm is our intuitive understanding of graphs.

FURTHER POSSIBLE IMPROVEMENTS

- The character recognition technique can be something optimized for number recognition, rather than just using pytesseract.
- Some kind of image cleaning/segmentation pre-processing can maybe be done to further improve the working efficiency of the algorithm.
- Code can be modified to work with graphs that have multiple curves of one colour.
- The colours of the background and the axes can be dynamically determined from the image.

THANK YOU