Graph Reverse Engineering

Utkarsh Garg | Uzair Bin Tariq

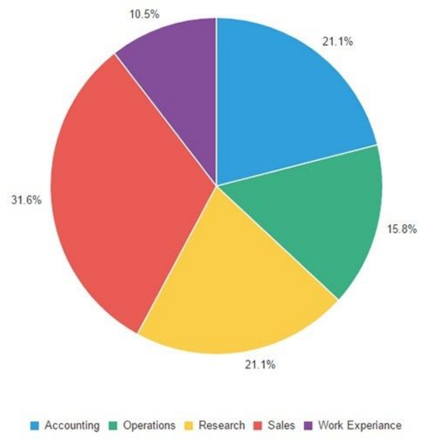
OVERVIEW

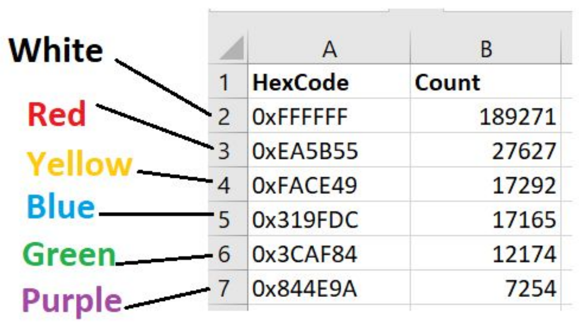
The implementation is broadly divided into two major phases. Our first phase primarily revolves around reversing the graph and making machines to understand the content of the chart. In simple words, extracting the contents of a chart, classifying their roles, and analyzing what is missing from the chart. This is indeed the most challenging part of the entire project. Once we are successful in making machines to understand the content of graphs, we will move to the next phase where we aim to beautify the chart and use text to speech services, to read out the content of the chart in a storytelling fashion. For this semester (CSE523), our goal would be to get significant results for the first phase of the project.

IMPLEMENTATION

- **Data Ink Ratio (JAVA)** | 10th March 2020

After thorough research and literature review, we started with the implementation of data-ink ratio extraction form a given chart. For this purpose, we used a naive, yet a robust approach to create a color frequency map. We initially implemented this task in JAVA but eventually re-implemented in Python to make our technology stack more consistent. Following the are results obtained:




White
Red
Yellow
Blue
Green
Purple

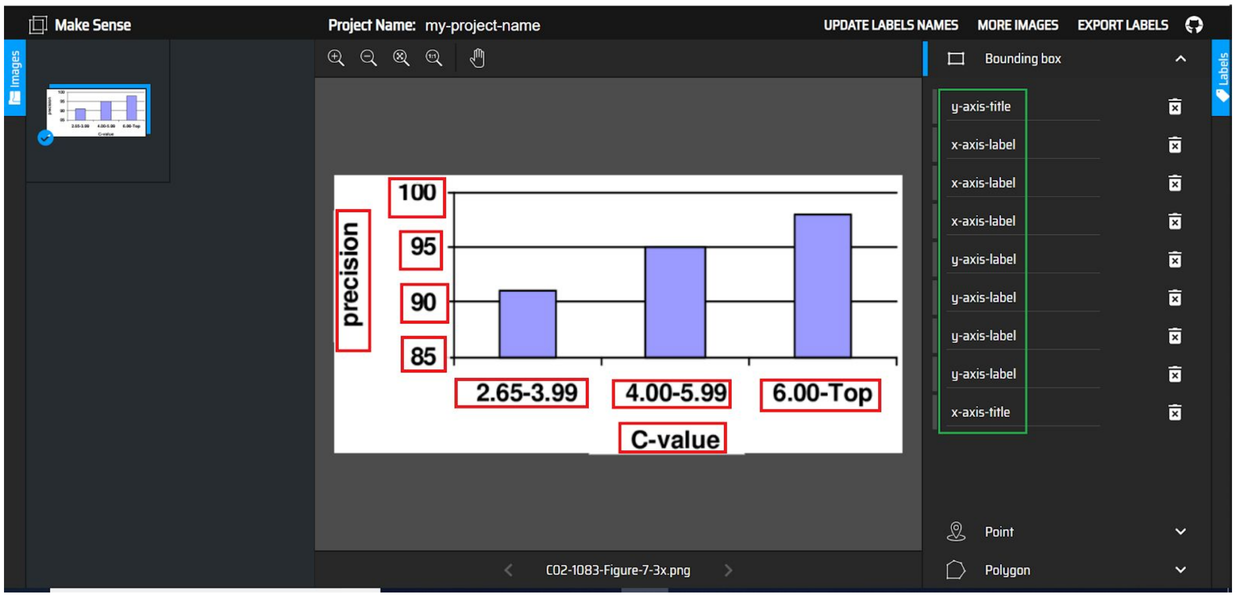
	A	B
1	HexCode	Count
2	0xFFFFFF	189271
3	0xEA5B55	27627
4	0xFACE49	17292
5	0x319FDC	17165
6	0x3CAF84	12174
7	0x844E9A	7254

- **Data Collection/Data Wrangling** | 31st March 2020

Initially, we had a labeled dataset consisting of features of different charts e.g. chart width, height, text coordinates, and text roles in a CSV format.

For more robust and accurate results, we extended this primary data source by adding additional charts from multiple sources and made use of the makesense.ai tool (https://www.makesense.ai/) to manually annotate and label elements of charts such as text bounding box coordinates, chart width, height, etc.

Following is an example of a single chart being manually annotated via makesence.ai tool

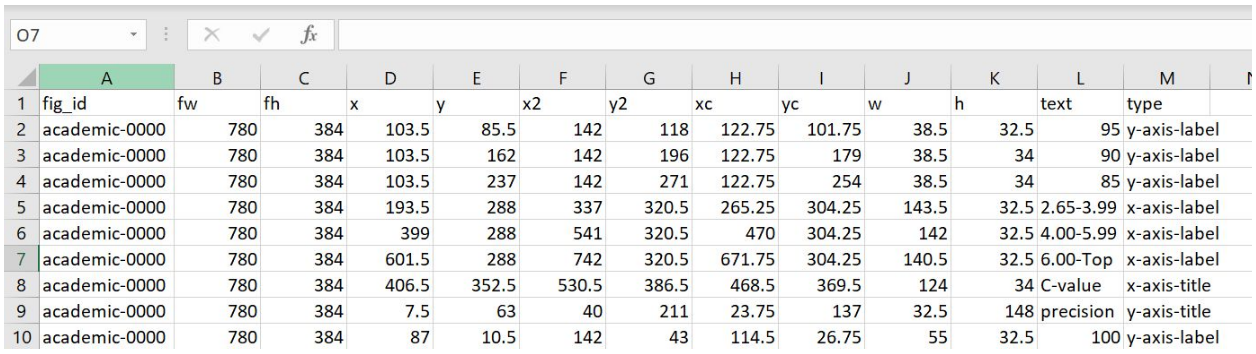


We also performed label encoding to encode textual data. Nevertheless, we also performed basic data wrangling techniques i.e. data cleaning, standardization, and data normalization before we moved on to the modeling phase for text-role classifiers.

Summary:

1. Data Collection
2. Data Augmentation
3. Data Cleaning
4. Label Encoding
5. Data Standardization
6. Data Normalization

Below are the features of a single chart in a CSV format. fw and fh are the width and height of the chart, x and y are top-left coordinates of a bounded box while x2 and y2 are lower right coordinates of a bounded box. Type is the true label or the text role in the chart.



O7

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	fig_id	fw	fh	x	y	x2	y2	xc	yc	w	h	text	type
2	academic-0000	780	384	103.5	85.5	142	118	122.75	101.75	38.5	32.5	95	y-axis-label
3	academic-0000	780	384	103.5	162	142	196	122.75	179	38.5	34	90	y-axis-label
4	academic-0000	780	384	103.5	237	142	271	122.75	254	38.5	34	85	y-axis-label
5	academic-0000	780	384	193.5	288	337	320.5	265.25	304.25	143.5	32.5	2.65-3.99	x-axis-label
6	academic-0000	780	384	399	288	541	320.5	470	304.25	142	32.5	4.00-5.99	x-axis-label
7	academic-0000	780	384	601.5	288	742	320.5	671.75	304.25	140.5	32.5	6.00-Top	x-axis-label
8	academic-0000	780	384	406.5	352.5	530.5	386.5	468.5	369.5	124	34	C-value	x-axis-title
9	academic-0000	780	384	7.5	63	40	211	23.75	137	32.5	148	precision	y-axis-title
10	academic-0000	780	384	87	10.5	142	43	114.5	26.75	55	32.5	100	y-axis-label

- **Modeling** | 10th April 2020

Once we had enough confidence in our dataset, we moved to generate sophisticated models for text-role classification in the given chart. We tried multiple models such as linear regression, K Nearest Neighbour (KNN), random forest, etc. Turned out that KNN worked best for the purpose of text-role classification.

We followed splitting the dataset into training and testing sets for different proportions. Following are the results obtained:

Training Set	Testing Set	MSE	Accuracy
60%	40%	0.27778	89.5141 %
70%	30%	0.22660	91.5199 %
90%	10%	0.22164	91.9525 %

- **Data Ink Ratio [Python]** 15th April 2020

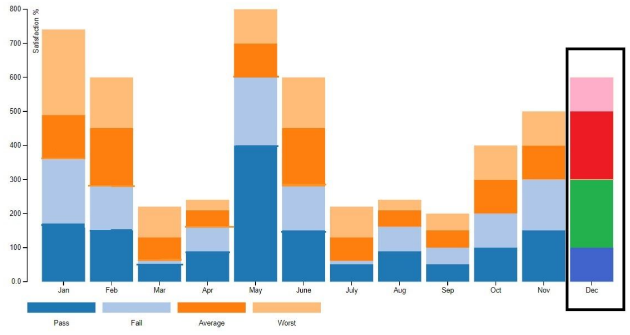
As discussed earlier, the data-ink ratio was re-implemented in Python for the technology stack consistency and so that we could finally integrate everything in a single standalone module in the shape of a web application using Flask.

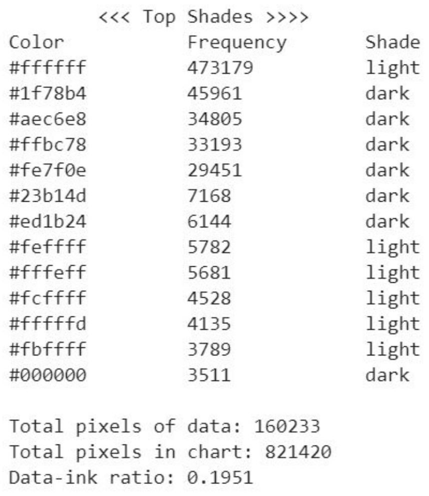
The data-ink ratio was calculated by the following formula:

Data-Ink Ratio = Data Pixels / Total Pixels

Calculating data pixels was a non-trivial task. We first created a color frequency map, then sorted colors according to frequencies in descending order. We also computed shades i.e. light or dark so that we could differentiate between chart background pixels and data pixels with an assumption that the color of the background is usually light.
We took the cumulative sum of the top 15 highest frequency unique and dark colors used in the chart. Following are the results obtained:

Below, left is a standard stacked histogram and the right is the output for data-ink ratio script.





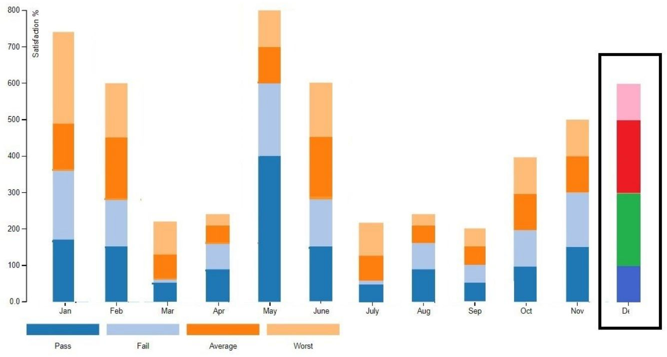
```
            <<< Top Shades >>>>
Color            Frequency        Shade
#ffffff          473179           light
#1f78b4          45961            dark
#aec6e8          34805            dark
#ffbc78          33193            dark
#fe7f0e          29451            dark
#23b14d          7168             dark
#ed1b24          6144             dark
#feffff          5782             light
#fffeff          5681             light
#fcffff          4528             light
#fffffd          4135             light
#fbffff          3789             light
#000000          3511             dark

Total pixels of data: 160233
Total pixels in chart: 821420
Data-ink ratio: 0.1951
```

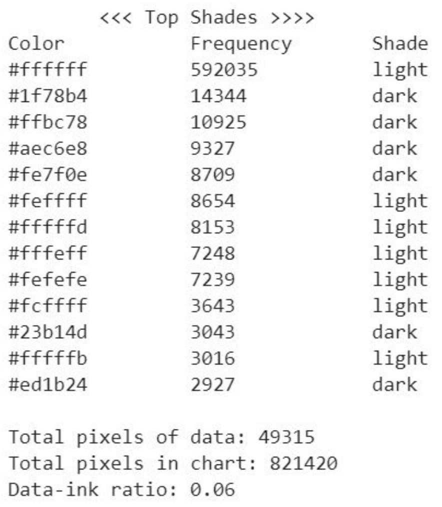
<u>Results:</u>

Total pixels of data = 160233
Total pixels in chart = 821420
Data-ink ratio = 0.1951

The data-ink ratio is greater than 0.15, therefore we can conclude that this is a well-designed chart with precise use of the space.

Now, let's take another version of the same chart, which represents the exact same data but is a bad version as the chart space is not utilized properly and hence the data-ink ratio would lower.




```
        <<< Top Shades >>>>
Color           Frequency       Shade
#ffffff         592035          light
#1f78b4         14344           dark
#ffbc78         10925           dark
#aec6e8         9327            dark
#fe7f0e         8709            dark
#feffff         8654            light
#fffffd         8153            light
#fffeff         7248            light
#fefefe         7239            light
#fcffff         3643            light
#23b14d         3043            dark
#fffffb         3016            light
#ed1b24         2927            dark

Total pixels of data: 49315
Total pixels in chart: 821420
Data-ink ratio: 0.06
```

<u>Results:</u>
Total pixels of data = 49315
Total pixels in chart = 821420
Data-ink ratio = 0.06

The data-ink ratio is less than 0.15, therefore we can conclude that this is a bad-designed chart, and space is not used wisely.

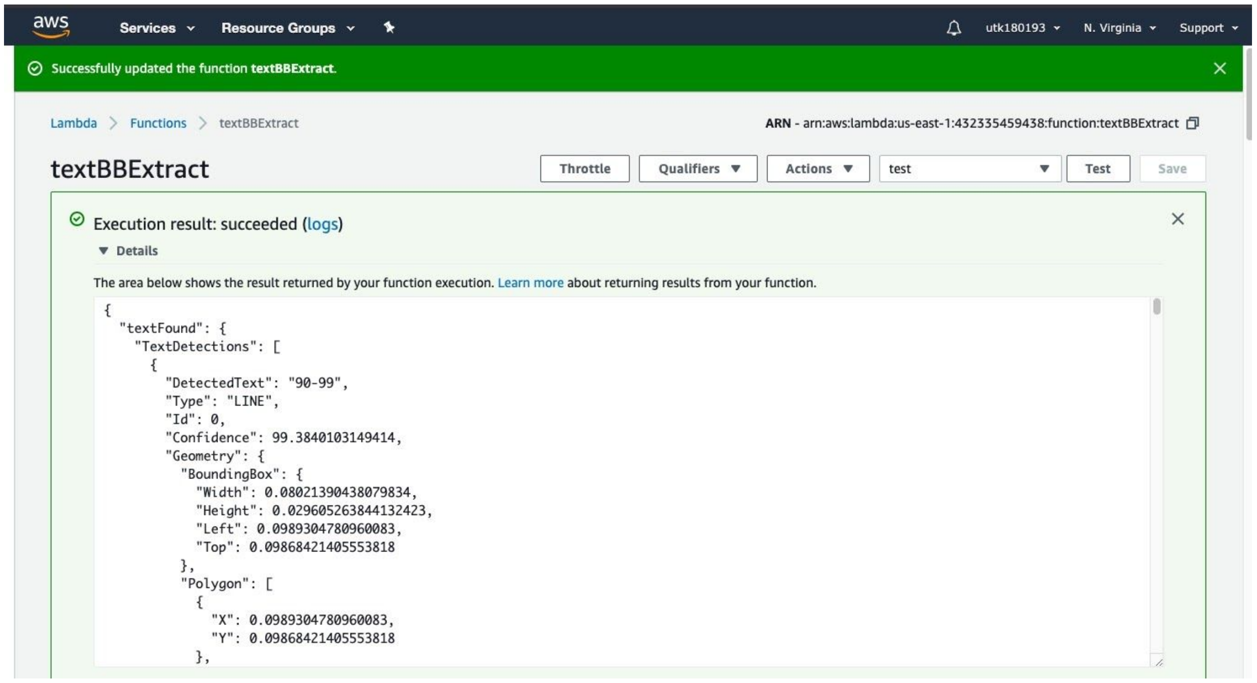
- **AWS Rekognition** | 20th April 2020

<u>Text detection over an image using Amazon Web Services.</u> *(see references)*

Configured AWS Lambda serverless function in Python 3.7, API Gateway (HTTP endpoint) together with S3, storage bucket for saving images uploaded images/charts. Performed extraction of text (labels/titles/legends etc.) from images/charts using AWS Rekognition API, to infer coordinates for various chart elements via bounding box.

These extracted bounding box coordinates would be leveraged in our pre-trained models for identifying the presence/absence of chart components and will also be utilized in graph scoring function (TBD).

Screenshot AWS Lambda Console:



Successfully updated the function **textBBExtract**.

ARN - arn:aws:lambda:us-east-1:432335459438:function:textBBExtract

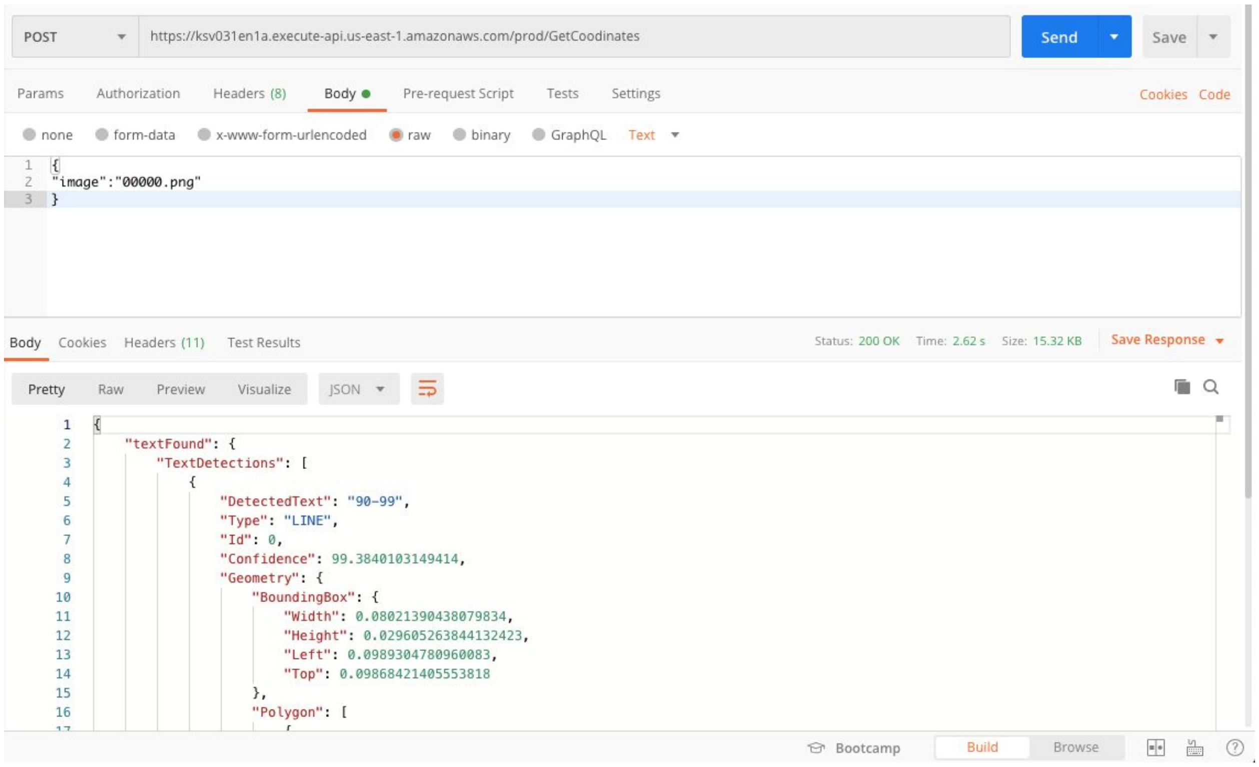
textBBExtract

Execution result: succeeded (logs)

The area below shows the result returned by your function execution. Learn more about returning results from your function.

```
{
  "textFound": {
    "TextDetections": [
      {
        "DetectedText": "90-99",
        "Type": "LINE",
        "Id": 0,
        "Confidence": 99.3840103149414,
        "Geometry": {
          "BoundingBox": {
            "Width": 0.08021390438079834,
            "Height": 0.029605263844132423,
            "Left": 0.0989304780960083,
            "Top": 0.09868421405553818
          },
          "Polygon": [
            {
              "X": 0.0989304780960083,
              "Y": 0.09868421405553818
            },
```

Testing Endpoint Over HTTP via POSTMAN :



POST https://ksv031en1a.execute-api.us-east-1.amazonaws.com/prod/GetCoodinates

```
{
"image":"00000.png"
}
```

Status: 200 OK Time: 2.62 s Size: 15.32 KB

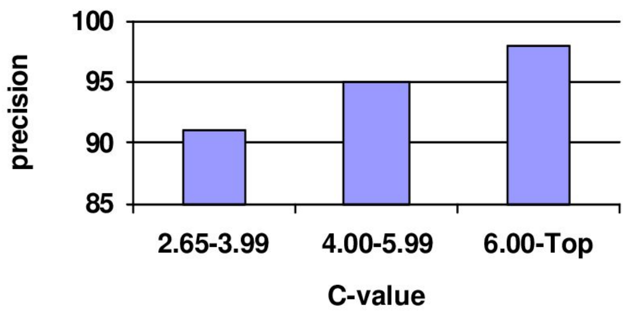
```
{
    "textFound": {
        "TextDetections": [
            {
                "DetectedText": "90-99",
                "Type": "LINE",
                "Id": 0,
                "Confidence": 99.3840103149414,
                "Geometry": {
                    "BoundingBox": {
                        "Width": 0.08021390438079834,
                        "Height": 0.029605263844132423,
                        "Left": 0.0989304780960083,
                        "Top": 0.09868421405553818
                    },
                    "Polygon": [
```

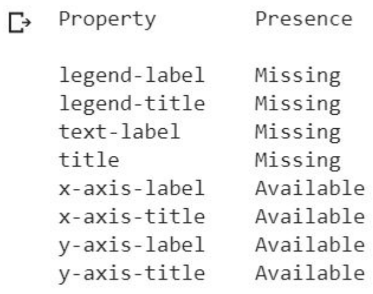
- **Testing** | 2nd May 2020

We plan to test in the following order:

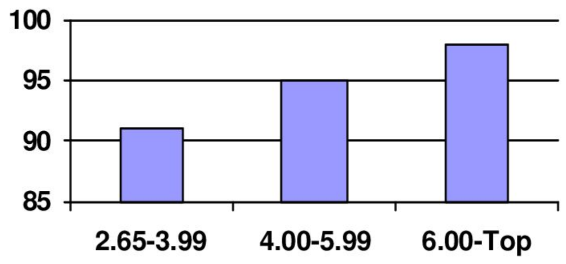
1. Get an input image from the user i.e. bar/line/pie charts.
2. Run AWS Rekognition to get bounding boxes, respective coordinates, and text in a given chart image.
3. Get the coordinates of the bounding boxes.
4. Save the response from AWS Rekognition to a file in JSON format.
5. Run our best model (KNN) on the output of AWS Rekognition (coordinates of bounding boxes) to perform text-role classification i.e. to see whether it is a label, title, or a legend or so.
6. Run data-ink ratio tests using Python locally.

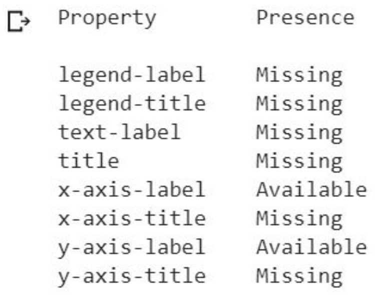
We've successfully executed the above-mentioned steps. Consider the following example for the outcomes. First, we run a chart with y-axis and x-axis titles, then for the same chart, we removed the x-axis and y-axis titles.





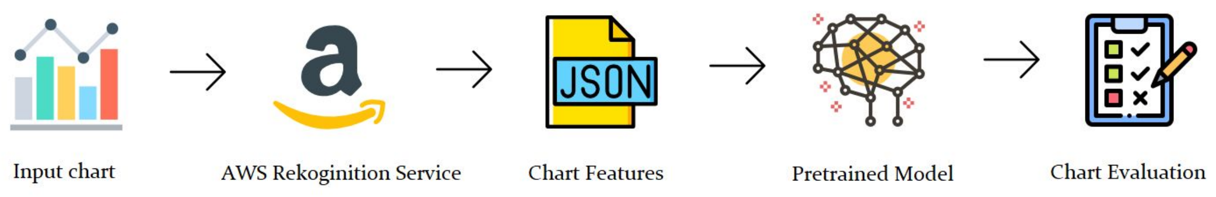
```
Property        Presence

legend-label    Missing
legend-title    Missing
text-label      Missing
title           Missing
x-axis-label    Available
x-axis-title    Available
y-axis-label    Available
y-axis-title    Available
```





```
Property        Presence

legend-label    Missing
legend-title    Missing
text-label      Missing
title           Missing
x-axis-label    Available
x-axis-title    Missing
y-axis-label    Available
y-axis-title    Missing
```

OVERALL ARCHITECTURE:



NEXT MILESTONE:

1. Implement a scoring function to score a chart as per Tufte principles.
2. Integrate data-ink ratio, text-role classifier, AWS lambda API for text bounding box into a single Flask based web-application.

LINKS:

1. Colab notebook for data-ink ratio: Click Here
2. Colab notebook for model training: Click Here
3. Image dataset: Click Here

Note: For now, we have 2 separate ipynb files at the moment, viz. ***model.ipynb*** and ***data_ink.ipynb***. Ultimately we will merge all relevant code into a single module with Flask as a backend for a complete end-to-end web application.

REFERENCES:

1. Base paper: http://idl.cs.washington.edu/papers/reverse-engineering-vis/
2. AWS Rekognition: https://console.aws.amazon.com/rekognition/home#/text-detection
3. Integrating Lambda with AWS API gateway: https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway-tutorial.html