CS732: Data Visualisation Assignment 2 Report

Sarthak Harne
IMT2020032
Sarthak.Harne@iiitb.ac.in

Sougandh Krishna K S
IMT2020120
Sougandh.Krishna@iiitb.ac.in

Monjoy Narayan Choudhury
IMT2020502
Monjoy.Choudhury@iiitb.ac.in

TASKS

Broadly, we have the following tasks:

Information Visualization

Under Information Visualization, there are three more tasks:

- Parallel Coordinates Plot
- Node-Link Diagram Plot
- Treemap Diagram

Scientific Visualization

Under Scientific Visualization, there are three more tasks:

- Contour Plot
- Quiver Plot
- Colormap

PROJECT STRUCTURE AND DIVISION OF WORK

The main directory has a report titled 'DV-Assignment-2-Fusion.pdf', which is the current file. The directory tree for the main project directory is as follows:

- DV-Assignment-2-Fusion.pdf
- Information Visualization
  - NodeLinkDiagram
    - Code
      - README
    - Images
  - ParallelCoordinatesPlot
    - Code
      - README
    - Images
  - TreeMap
    - Code
      - README
    - Images
- Scientific Visualization
  - ColorMap
    - Code
      - README
    - Images
  - ContourPlot
    - Code
      - README
    - Images
  - QuiverPlot
    - Code
      - README
    - Images

The details regarding the Code and Images directory can be found in the respective visualizations' section. The division of work is as follows:

- Information Visualization
  - Node Link Diagram - Monjoy Narayan Choudhury
  - Parallel Coordinates Plot - Sarthak Harne
  - Tree Map - Sougandh Krishna KS
- Scientific Visualization
  - Quiver Plot - Monjoy Narayan Choudhury
  - Contour Plot - Sarthak Harne
  - Color Map - Sougandh Krishna KS

INFORMATION VISUALIZATION

Node Link Diagram

Dataset: For this task, we have chosen the 'A Song of Ice and Fire' dataset. This dataset contains a fictional social network of the fantasy novels George RR Martin wrote. An edge denotes that two characters are mentioned within fifteen words of each other while the weights of the edge denote the number of such appearances. The dataset version used for the experiments can be found in the link here. To be specific we use the dataset files labeled: *asoiaf-all-nodes.csv* and *asoiaf-all-edges.csv* that consist of the node and connections information respectively. The dataset consists of 794 nodes and 2823 edges between them. The graph structure is undirected in nature.

Preprocessing steps performed: We leverage Gephi's various statistical models to gain more insights into the data. The unprocessed data upon visualized by directly loading can be seen in Figure 1.

Initially, we start our study by computing the average degree using Gephi's statistic tools which evaluates to about 7.093. The degree distribution can be seen in Figure 2. To make the data readable we go ahead with a degree filtration method to cut down the number of nodes in some of the visualization that we will present next. While for the others we use the filtration to filter out the labels only.

Channels used for Node: Following are the channels that we tried to leverage for the Nodes to express more about the dataset.

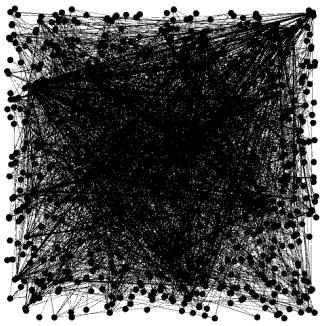

Fig. 1: Initial Look upon loading the dataset

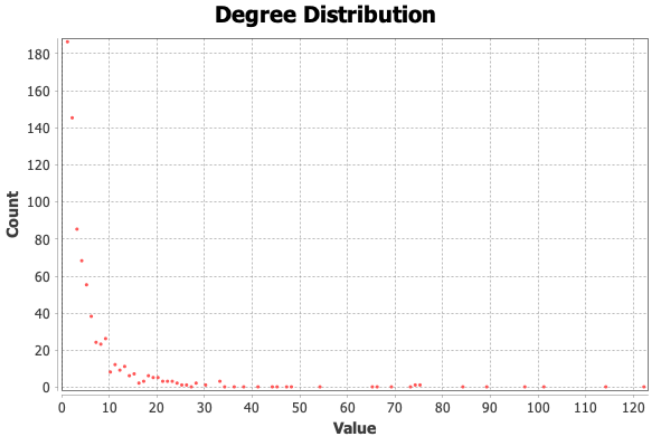

Fig. 2: Degree distribution plot generated by Gephi

1) Size: The size of the nodes are varied based on the degree of the node. Since this is an undirected graph the notion of in-degree and out-degree did not make much sense and we decided to use degrees to view some of the nodes larger than the others. This is to bring in some notion of ”centrality” or important characters. Since we had access to the label on this version of the dataset we were able to verify empirically our intuition as the major characters of the novel had the highest degrees as it is expected to be with the node labeled Tyrion Lannister being the largest. The initial idea of our implementation with size can be seen in Figure 3
2) Node Color: For node color, we experimented with the partition option for visualization parameters in Gephi and we partitioned the nodes by ”Modularity Class”. For using this feature, we first had to compute the Modularity detection algorithm as proposed by [1]. The strength of the network graph's separation into clusters is measured by modularity. High modularity indicates that connections between nodes in the same cluster are dense, with minimal connections to nodes in other clusters. The modularity score obtained is 0.56 for about 11 communities. We decided to color code these where the Geiphi palette had distinguishable colors for the larger 8 communities. Upon partitioning based on modularity, we were able to see communities that made empirical sense based on the dataset used. We will discuss more on the communities discovered and the correctness in the next section.
3) Edge Color: We used the edge color to denote the edge weight. For this, we used a simple single base color gradient-based colormap that suggested how strongly related 2 nodes were. Here we noticed that characters that are siblings for eg: The Lannisters (Jamie, Tyrion, and Cersei) who were siblings had distinguishable dark-colored edges citing that they had a lot of co-occurrence in sentences together due to the heavy exchange of dialogues between them. The initial experimentation of this idea can be seen in Figure 4

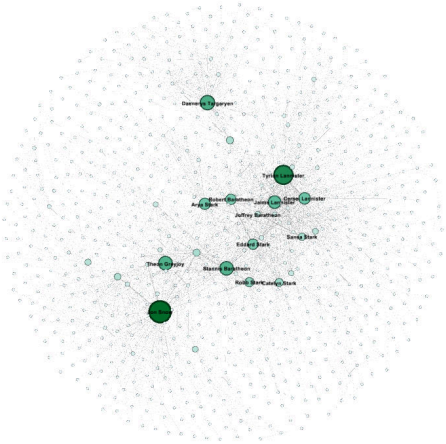

Fig. 3: Identifying important nodes with the help of size

Layout Algorithms Experimented:

- **Fruchterman Reingold**: The Fruchterman Reingold force-directed layout in Gephi provides us with the following parameters:
  1) Area: Graph size area where they prescribe you to consider the value 1000 for 100 nodes. We used 700 as per the prescribed guide in the GUI itself.
  2) Gravity: This is the force in the ”force-directed” nature. This is done to make sure that over-dispersion of disconnected nodes doesn't take place. We have decided to use 4.0 as the value based on the visual output.
  3) Speed: Convergence speed of the algorithm. This is actually a tradeoff parameter where the precision of the layout is compared with the convergence of the algorithm.
  4) Since this is not an auto-terminating algorithm as

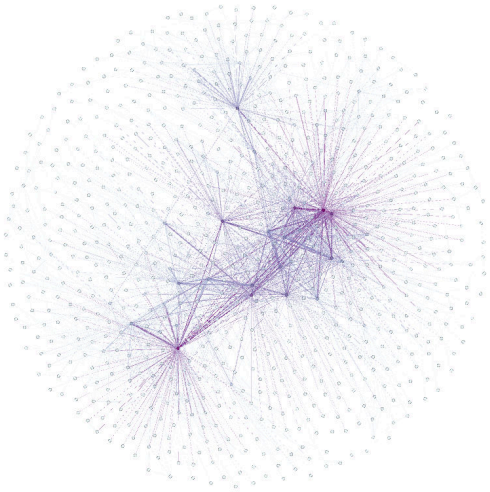

Fig. 4: Coloring edges based on weights

such. We ran the algorithm for about 2800 iterations till the layout stabilized with nodes not displacing much.

The output of this algorithm along with all the channel modifications as discussed above can be seen in Figure 5 To reduce some of the nodes we apply a degree filter with

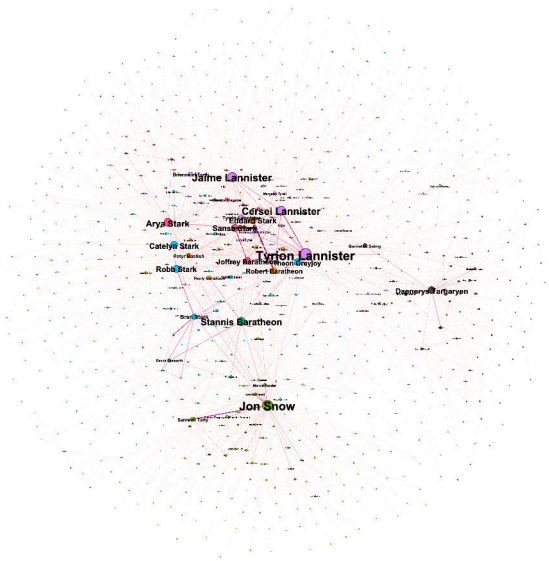

Fig. 5: Fruchterman Reingold Layout for the given dataset.

the criteria of removing all nodes from the visualization with a degree less than 10. The output of that operation gives us a much more readable visualization that can be seen in Figure 6.

- **Force Atlas**: Jacomy et al. created the Force Atlas layout [2] algorithm while working on Gephi. This iterative

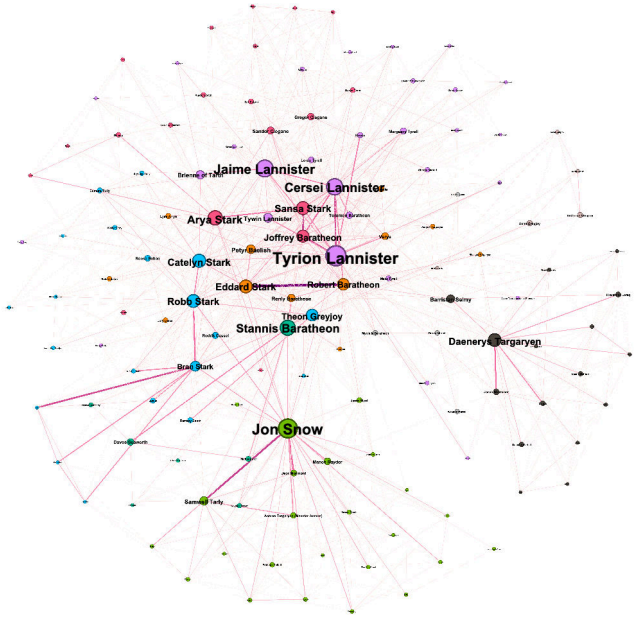

Fig. 6: Fruchterman Reingold Layout for the given dataset. Nodes with degrees less than 10 are not shown. The Node labels are sized based on the value of the degree and shown for all nodes for drawing insights about character relations.

method operates as follows:

1) Separate the nodes with the most connections (the hubs) by pushing them apart from one other.
2) Pull the nodes that are linked to those hubs together.

As a result of numerous iterations, the most influential nodes are spread across the representation space, while lesser nodes related to them are drawn towards those hubs, forming communities. In Gephi, the following parameters are present and used for creating the required layout:

1) Intertia: Node speed at each iteration. Set to 0.1
2) Repulsion Strength: It is a measure of rejection of closeness by each of the node. Set to 10,000.
3) Attraction Strength: Set to 50.
4) Autostabilise function set to on. This allows unstable nodes to be frozen.
5) AutoStab Strength and sensibility are parameters associated with above option and default values of 80,0.2 are taken.
6) Gravity: Similar to Fructherman Reingold Gravity. Set to 30.0
7) Attraction Distribution switched off to prevent nodes going far due to distribution of forces uniformly over the nodes.
8) Adjust by Sizes: an added advantage over Fructherman Reingold that allows the ForceAtlas algorithm to take care of node size requirements based on criteria (in our case degree).

9) Speed: Same as Fructherman Reingold. Set to default values of 1.

The output yielded by this algorithm can be seen in Figure 7 As you can already notice the layout for all communities partitioned using modularity is better visualized for most of the nodes and is easier to notice the natural clustering of data using visualization. Upon filtering the nodes based on the same criteria for the previous layout, we get the output as seen in Figure 8

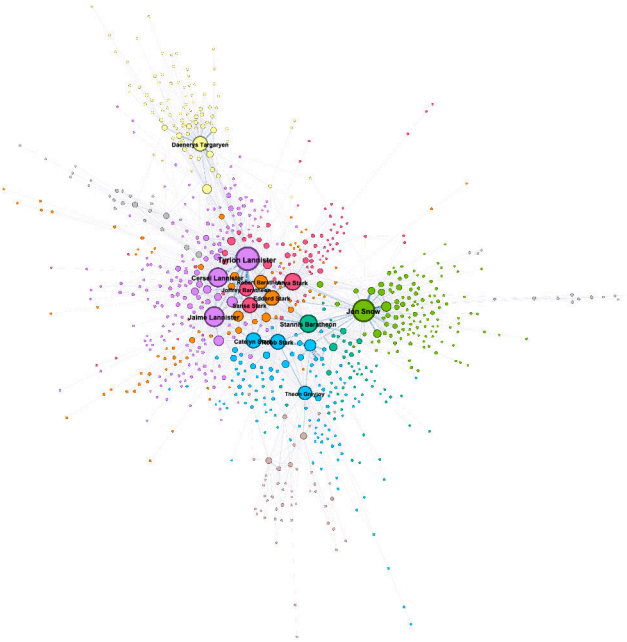

Fig. 7: Force Atlas layout output for all nodes.

- **OpenOrd** [3]: It takes in an undirected weighted graph and aims to recognize clusters more effectively. It is highly parallelizable and has a sure-shot termination. The algorithm is based on Frutcherman-Reingold and operates with a predetermined number of iterations that are regulated by a simulated annealing type schedule (liquid, expansion, cool-down, crunch, and simmer). To allow clusters to separate, long edges are clipped. The following are the parameters available in Gephi for this algorithm:
  1) Stages which consist of the simulated annealing options mentioned above. We have let Gephi default values be used for the experiment.
  2) Edge Cut: This ranges from 0 to 1 where using 0 is the same as using the Fructherman Reingold algorithm while values close to one will result in a better visual clustering of nodes. We set the value to 0.9.
  3) Num Threads: is a parameter useful for parallelizing the algorithm. This is set based on the CPU of the machine being used and preferably not altered.

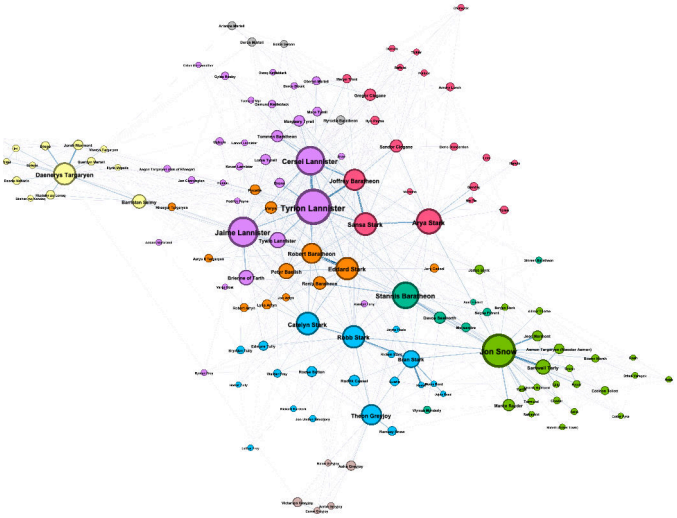

Fig. 8: Force Atlas layout output. Nodes with degrees less than 10 are not shown. The Node labels are sized based on the value of the degree and shown for all nodes for drawing insights about character relations.

  4) Num iterations: This controls the density of nodes of the graph. The more the number the less dense the graph becomes and convergence time increases. We have taken the value 1500 (twice the default value of 750).
  5) Fixed time: it measures the time for which the node is not in motion. The default value is used for this.
  6) Random seed: the result depends on starting with a random layout. To produce consistent data over machines this is used to define the seed. The default value as per Gephi is used.

  The output of this algorithm can be seen in Figure 9 and Figure 10 for the filtered version. The layout brings nodes of similar type closer to each other, however, the clustering is not visually effective as seen in Force Atlas. Also, the nodes of the same community collapse into each other making nodes unreadable.
- **Radial Axis Layout**: It groups nodes and draws the groups in axes (or spars) radiating outwards from a central circle. Groups are generated using a metric (degree, betweenness centrality...) or an attribute. This is not a force-directed layout algorithm and provides a different perspective to the data being visualized. Some of the parameters explored in this algorithm are:
  1) Scaling Width: this provides with distance between nodes. Set to 1.2.
  2) Node Size: For resizing nodes, the base size of the node. Set as 2.

Fig. 9: OpenOrd layout for entire data.

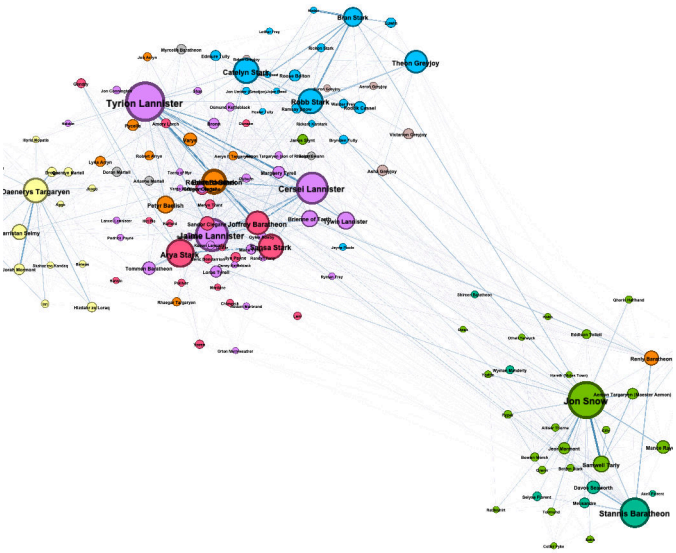

Fig. 10: OpenOrd layout for the filtered data.

3) Group Nodes by: feature to group nodes based on some parameter. For our experiments, it's the modularity result.
4) Order Nodes: Arranging nodes from center to radially out based on a parameter. We have used the degree parameter.

The output for both the complete data and the filtered data can be seen in Figure 11 and Figure 12.

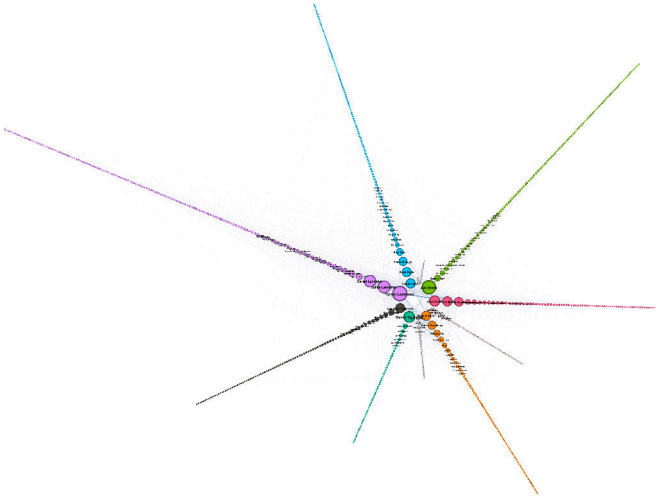

Fig. 11: RadialAxis layout for entire data.

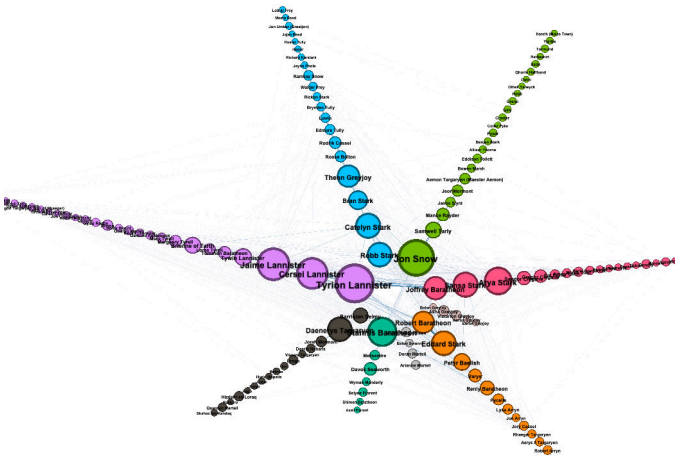

Fig. 12: RadialAxis layout for the filtered data.

The groups get highlighted pretty well in this layout, however, the edge crossings increase by a large number, and the sense of edges between elements from different communities becomes tougher to figure out and the notion diminishes. Among all the tried algorithms, ForceAtlas provides a good balance between community detection and edge connectivity visualization.

Inference: Using this visualization one is expected to achieve the inference of the following tasks:

- Identify the important characters of the book. As for important characters we consider that they have the most amount of dialogue and with most different characters.
- Identify communities in the data. This will let us understand how certain characters are related in the story.
- Identify the strength of relationships and alliances between various characters based on the communities formed. As we can clearly see certain groups of characters are associated with certain main characters and in the story itself it can be seen that they were part of the same alliance.

Resources used: The following resources have been helpful in coming up with the visualization:

1) Gephi Tutorial by Volodymyr Miz on January 5, 2020, accessed from here.
2) Official Gephi tutorial on layout access on November 10, 2023, from here.
3) Gephi Video tutorial from YouTube by Jengol Beck accessed on November 6,2023.

Parallel Coordinates Plot

Dataset and Preprocessing: The same dataset as Assignment-1 i.e. Global YouTube Statistics was used for the Parallel Coordinates Plot (PCP).

For Preprocessing, a new field called 'Days Since Created' is added to the dataset. This is done using the 'Created Year', 'Created Month', and 'Created Date' fields.

Next, the top 7 Countries by number of channels are selected. Without this, the plot becomes very visually cluttered. Also, only the top 10 channels from each country are chosen for the same reason. Fig 13 shows the case when filters are not applied. Although some conclusions regarding the number of channels from a particular country can be drawn, the clutter prevents further inferences. This problem persists even when other colour maps are chosen. To prevent cluttering, the

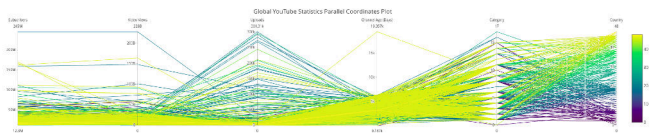

Fig. 13: The Plot is very cluttered when rows are not filtered

following constraints are placed:

- Only the top 7 countries according to the number of channels are chosen.
- Only the channels with a Country Rank less than or equal to 10 are chosen. This limits to at max 10 channels from one particular Country.

This gives us a better and uncluttered initial PCP layout. Fig. 14 shows the initial layout of the PCP after the filtering. The following attributes of the dataset are plotted:

- Subscribers: The number of subscribers the channel has

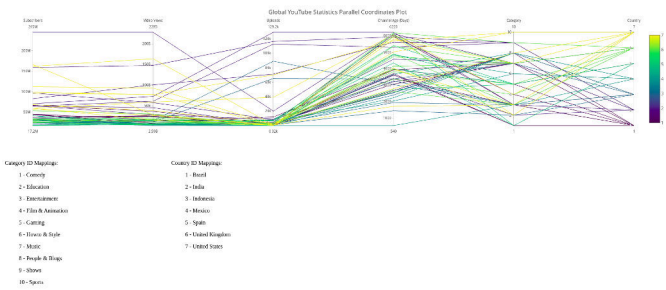

Fig. 14: Initial PCP layout after filtering

- Video Views: The total number of views the channels has
- Uploads: The total number of uploads done by the channel
- Channel Age (Days): The Days it has been since the channel was created. Calculated as mentioned before.
- Category: The category assigned to the channel by YouTube. Originally strings, used a 1-indexed label encoding to plot.
- Country: The Country the channel belongs to. Originally strings, used a 1-indexed label encoding to plot.

The mappings for the Country and Category attributes are mentioned with the plot as can be seen in Fig. 14

Interactions: The plot is interactive to aid with the process of Visual Analytics. Axis brushing and Axis reordering can be used as is convenient for the user.

Axis Brushing can be used to filter out and select only certain ranges of attributes at once. This is done via clicking and dragging the mouse over the desired range(s) on the axis of any attribute. For example, Fig. 15 shows how only certain Countries and Channel Ages can be selected using Axis Brushing. Note that, we can have multiple range selections for a single attribute as done in the case of the Country attribute.

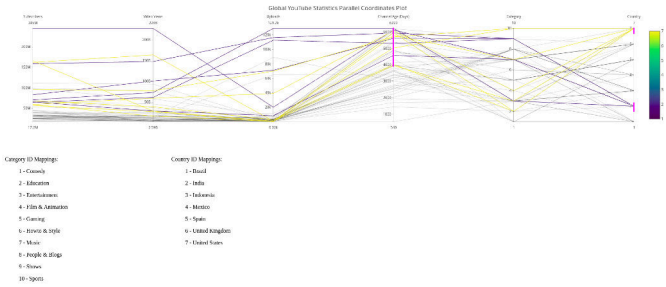

Fig. 15: Using axis brushing to select specific Countries and Channel Age

Axis reordering can also be used to observe direct correlations between two attributes. This is done via clicking and dragging the name of the attribute and moving it as desired. For example, in Fig. 16 the relation between Channel Age (Days) and Subscribers is more visually apparent after the axis reordering is done as comapred to Fig. 14

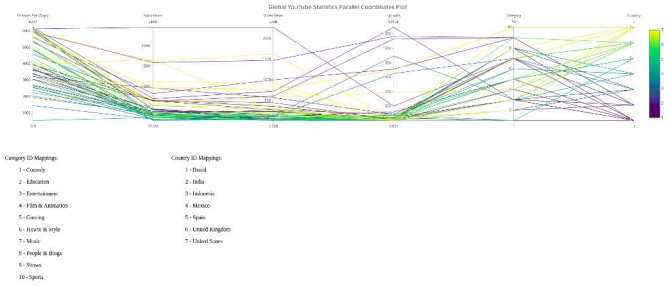

Fig. 16: Using axis reordering to study the correlation between two attributes

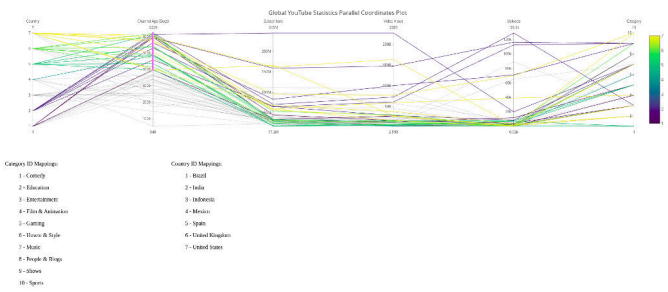

Fig. 17: Older Channels and related attributes

Implementation: The data as processed in Assignment-1 is first obtained from Tableau and is further processed as mentioned before using Python. This data is then exported as a CSV to be utilized later on.

Plotly along with D3.js are used to plot the Parallel Coordinates Plot as a browser implementation. The dataset is first imported. Next, the Colorscale is chosen and is set according to the Country attribute of the data item. Next, all the other attributes are added to the plot before displaying the plot along with the ID mappings for Category and Country attributes.

Inference: Using the interactions, the following inferences can be drawn using the PCP:

- The outliers can be clearly spotted in 14 in the Subscribers, Video Views and Uploads attributes as lines which are not clumped together. These are channels with more than 50M subscribers, more than 50B Video Views and more than 20k Uploads.
- After brushing the older channels, it is apparent from Fig. 17 that Older Channels tend to have more Subscribers, Video Views and Uploads. Most of the channels India, Spain, United Kingdom and United States are older channels. On the contrary, after brushing the newer channels, it is apparent from Fig. 18 that Indonesia and Mexico contribute the most to these. They also have lesser Subscribers, Video Views and Uploads.
- Brushing to select the channels with very high number of Subscribers reveals in Fig. 19 that the channels are more than 10 years old and originate mostly from the United States and India, with one coming from United Kingdom and Brazil each.

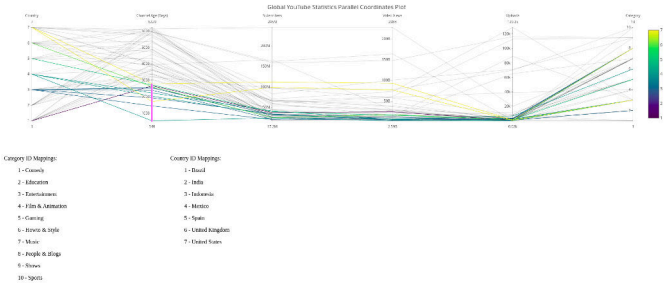

Fig. 18: Newer Channels and related attributes

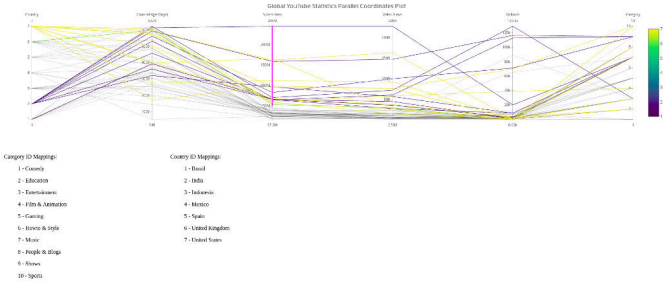

Fig. 19: Channels with High Subscribers and related channels

- Brushing to select the channels with very high number of Video also reveals in Fig. 20 that the channels are more than 10 years old. But in this case, all the channels either come from the United States or India.

Tree Map

Dataset: The same dataset as Assignment-1 i.e. Global YouTube Statistics was used for the plots

Preprocessing: Tableau handled NULL values or Empty cells in .csv files independently. But here I had to remove NULL values, so did preprocessing and created a new CSV file YT2.csv. So for columns with a float value, I replaced it with 0, and for columns with sting, I replaced null with Unknown.

Interaction: Treemap plots are very much interactive.

- You can hover over a node and it displays more data related to that node.

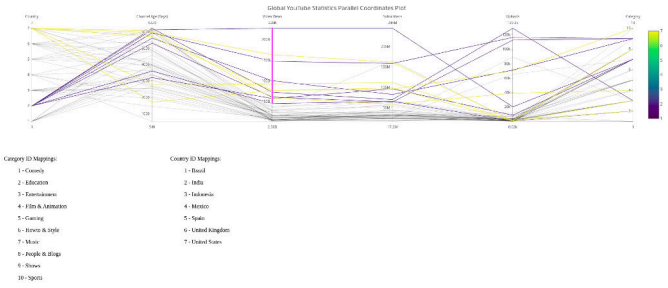

Fig. 20: Channels with High Video Views and related channels

- If you click on a node it maximizes that node to full screen so that you can easily read whatever is printed inside that rectangle.
- In case of nested treemaps, It will take you inside another treemap.

Tools Used: HTML, js (Papa.parse to read .csv files and Plotly to plot)

Visualisations: First visualisation, Fig 21 was to plot a treemap with Youtubers as nodes and their subscriber count determining the size of the rectangle and its color gradient. This was a pretty simple plot and I had restricted it to the top 30 as the dataset had 900+ YouTubers and the map looked very congested.

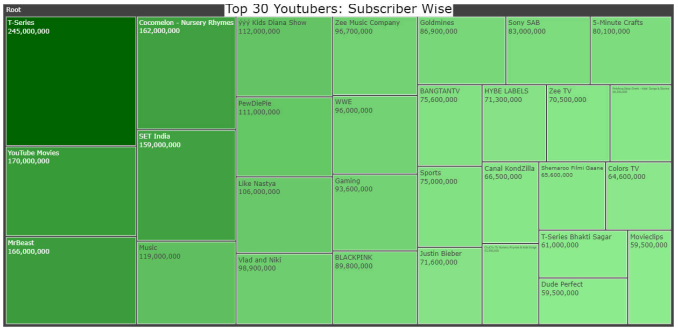

Fig. 21: Top 30 Youtbers, Subscribers Wise

In the next visualization, Fig 22 2 variables were taken to plot the treemap whereas there was only one in the previous visualization. Here, The sizes of rectangles were determined by the number of subscribers the YouTuber has and the gradient of color was determined by their highest annual income.

In Assignment 1, we have seen that all the income-based columns are very much proportional and hence it doesn't matter which income-based column you consider for the plot. The difference between using 1 variable and 2 variables can be clearly seen when you compare both the treemaps.

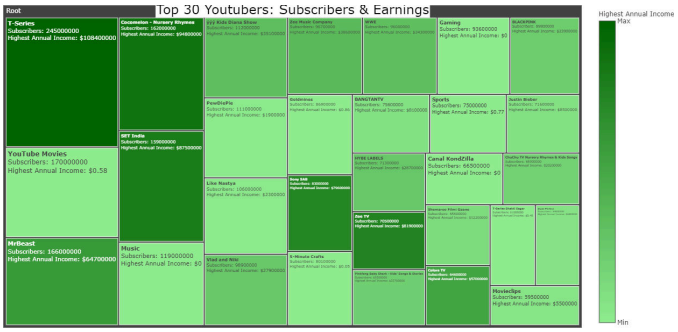

Fig. 22: Top 30 Youtbers, Subscriber Wise with Income determining gradient

Now in Visualisation 3 (Fig 23) and 4(Fig 24), I have taken the same parameters as done in the first two visualizations but with Video views as a column instead of subscribers. Here, you can see a linear relation if you observe the map carefully. We have inferred the same in the first assignment.

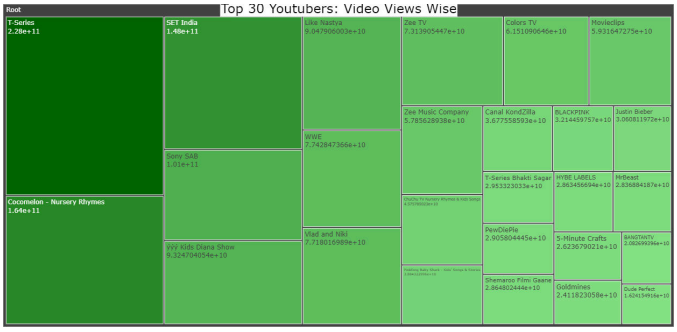

Fig. 23: Top 30 Youtbers, Views Wise

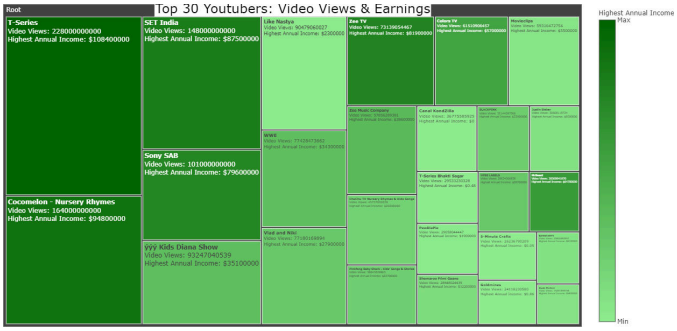

Fig. 24: Top 30 Youtbers, Views Wise with Income determining gradient

Now in the next visualization, Fig 25 I have implemented nested treemaps. First, treemap with category-wise distribution, and then if you click on any of the categories, It will open up an inner treemap where you can see the Youtubers with the most subscribers in that category.

Figure 26 shows the map that you will see once you click on the entertainment node. Figure 27 shows the inner tree map that you will see once you click on the education node. Here in the inner treemap, I used only one variable, where the gradient will also be determined by the number of subscribers.

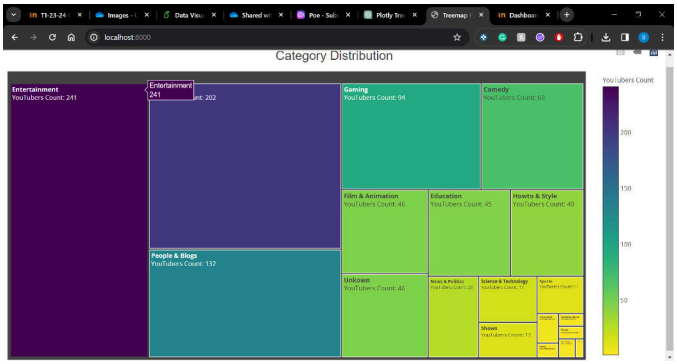

Fig. 25: Category Wise Distribution

Fig. 26: Top Youtubers in Entertainment Category

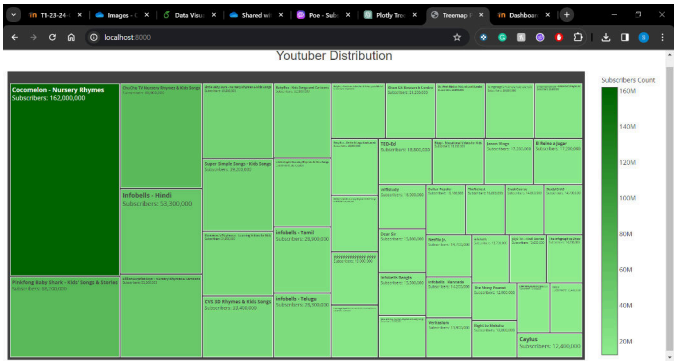

Fig. 27: Top Youtubers in Education Category

Inference:

- The Entertainment category boasts the highest number of content creators on YouTube(As per the given dataset).
- T-Series tops the list of YouTubers with the most subscribers, Most views, and also the channel that generates the highest revenue.
- When considering top subscribers within their respective categories, noteworthy names include Cocomelon leading in Education, and Mr. Beast reigning supreme in the Entertainment category.

Resources:

1) Plotly Treemap Tutorial here.
2) Treemaps in javascript here

SCIENTIFIC VISUALIZATION

Quiver Plot

Dataset Description: We use the OSCAT dataset for this task. This dataset provides information on the ocean surface vector wind (OSVW) in support of research and operational weather forecasting. The dataset consists of 2 components to be used: The Meridional and Zonal values. This is similar to the 2 orthogonal components that would create the resultant flow vector. However, this data is only present for the year 2013-2014 due to the anomaly suffered in February 2014. We select the first quarter of 2013 (inclusive of 1^st April). We select the following dates for our analysis.

- $1^{st} January$
- $15^{th} January$
- $1^{st} February$
- $15^{th} February$
- $1^{st} March$
- $15^{th} March$
- $1^{st} April$

Preprocessing: In the OSCAT dataset, luckily we didn't face any missing data similar to the one in the AMSR dataset. Our preprocessing steps involved mostly understanding the structure of the file that comes with the data. We stripped away all the metadata from the first 5 lines and stored it to use as the title of our visualization. From the metadata itself, we find out the exact value for the "BAD_FLAG", which basically signals the lack of data either due to the fact that the grid point corresponds to land or the data is corrupted. We have replaced these values with "NaN" with the help of numpy and matplotlib automatically rejects plotting these points. At the 6th line of the dataset, we find out all the longitudes present in the dataset. This line is followed by the latitudes present as the first item in the line followed by the corresponding values for all the latitude, and longitude grid points. The dataset consists of about 721 by 360 (Longitude-Latitude) points. Since visualizing the entire world would not yield detailed inferences of a region, we take the part of the Indian subcontinent with latitude ranging from 0 to 20 and longitude ranging from 68 to 100.

Implementation: For our implementation, we used Cartopy and Matplotlib along with PIL for gif animations and AnimationWriters from matplotlib for mp4 creation. We implement a utils package that handles loading the data and also attempts to create slices of data based on the coordinates we supply. Due to very less grid points, we didn't notice much difference in performance and didn't get much time to test the slicing extensively, hence we avoided usage of it. All the implementation of data loading and sliding can be found in the *utils.py* in the codebase for quiver plots. Regarding the plotting of data matplotlib API for quivers pretty much handles all the experiments. However, we tried out various parameters present to improve our visualization and also implemented a color map for the arrows. For streamlines, we use the maplotlib streamlines API for an initial visualization and also try to integrate color mapping.

Generation of plots: As discussed above in implementation we try to generate visualizations and improve them progressively. Initially, we try to get the API running and visualizing data. Initially, we try to generate the quiver plots with arrows of the same length. For this using the Zonal and Meridional values we need to compute its magnitude and individually normalise the values. Upon doing this, we get the following output as seen in Figure 28

To make any sense of magnitude for this plot, we necessarily need a colormap for it to make sense of it. We use the viridis colormap to denote the values as it is perceptually uniform as discussed in class. The output can be seen in Figure 29 We try to now include the length as an additional channel to denote the magnitude. To control the drawing of the arrows

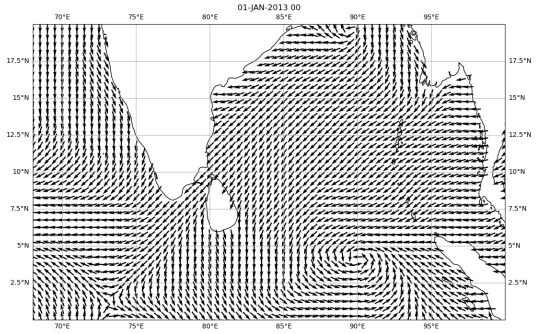

Fig. 28: Quiver Plots made with same length arrows.

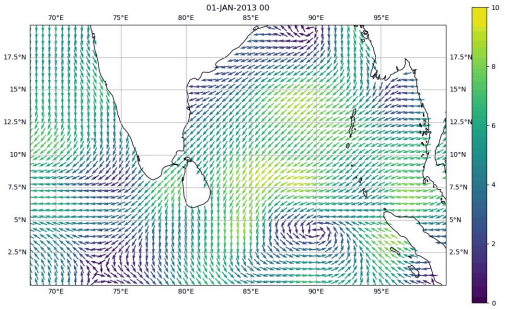

Fig. 29: Quiver Plots made with same length arrows and with colormap for denoting magnitude.

matplotlib uses the angles, pivot, and scale_units. The angles define what to use to define the angle of the vector for which we use the uv macro that denotes the angle to be computed based on the zonal and the meridional value while scale_units decide the units of the arrow to be drawn on the screen. There are two modes that we try ”heights” and ”widths”. The output for this can be seen in Figure 30 and 31 for widths and heights respectively.

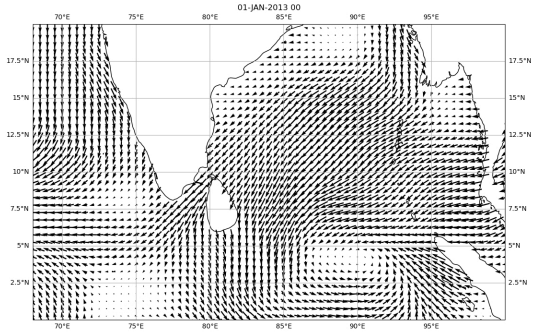

Fig. 30: Quiver Plots made using widths construction.

We noticed that ”heights” construction was clearly less

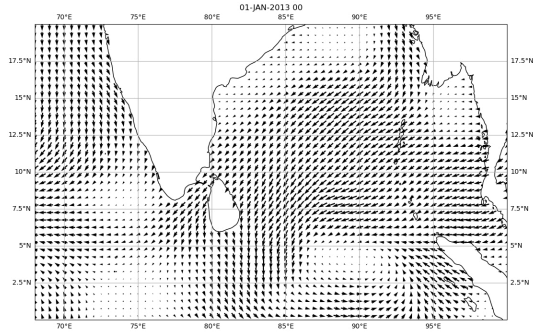

Fig. 31: Quiver Plots made using heights construction.

cluttered and readable and hence we used it as our choice. The visualization with colormap can be seen in Figure 32.

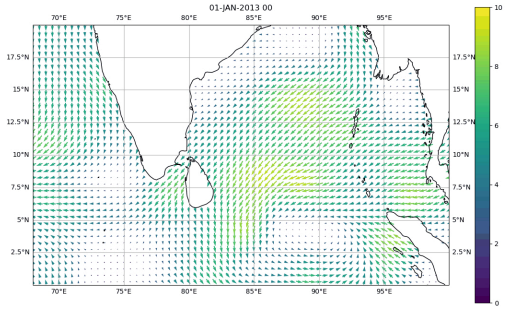

Fig. 32: Final Quiver Plot with Viridis Colormap.

Grid Sampling Technique: We try two more point sampling techniques from the grid as it impact the appearance and accuracy of quiver plots. Alongside this we try to also plot the entire world’s quiver plot. The 2 major techniques used are discussed as follows:

1) Subsample Grid Sampling (’subsample’): This technique involves subsampling the grid by taking every k^{th} point along both latitude and longitude axes. The quiver plot is generated using a reduced set of data points, providing a sparser visualization of the vector field. The output can be seen for the current part of the map selected in Figure 33 where k = 2. We use this to see the entire world map also in Figure 34 where k = 16.
2) Interpolation Grid Sampling (’interp’): This technique involves interpolating the vector field values to a new grid of higher resolution. The quiver plot is generated using interpolated values on a denser grid, providing a smoother representation of the vector field. The output for the slice of the map taken can be seen in Figure 35. We use griddata from SciPy to interpolate the vector field components (Zonal and Meridional values) from the original grid to this new grid.

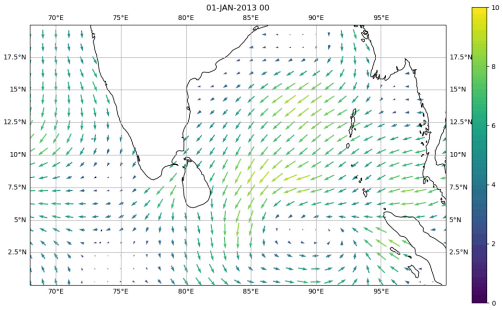

Fig. 33: Quiver Plot using subsampling with k = 2.

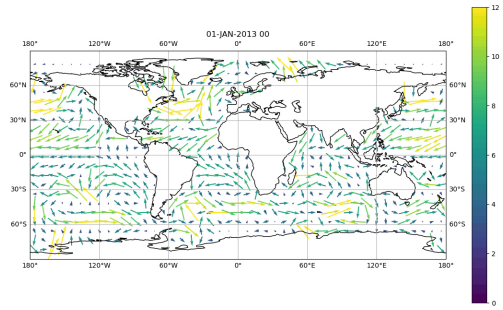

Fig. 34: Quiver Plot for the entire worldmap using subsampling with k = 16.

Streamlines: To additionally explore the data we try plotting streamlines using the matplotlib API. Even here we try to put in a colormap to make the data and trends more understandable. The output of streamlines without and with colormap can be seen in Figure 36 and Figure 37 respectively.

One thing to note is that the time taken to build a streamline plot is much higher than building an arrow plot.

Inference: Following are some of the tasks that are expected to be achieved with the help of the visualization present:

- Identify regions that are affected by high wind speeds.
- Identify directions along which the overall wind currents are flowing.
- Identify the change in wind currents and a corresponding shift in current progressively in every fortnight.

Contour Plot

Dataset: The dataset chosen was the Ocean Sea Surface Temperature dataset. The range chosen was the first quarter of 2023. The following dates are chosen for the 9 plots:

- $1^{st} January$
- $11^{th} January$
- $21^{st} January$
- $1^{st} February$
- $11^{th} February$
- $21^{st} February$

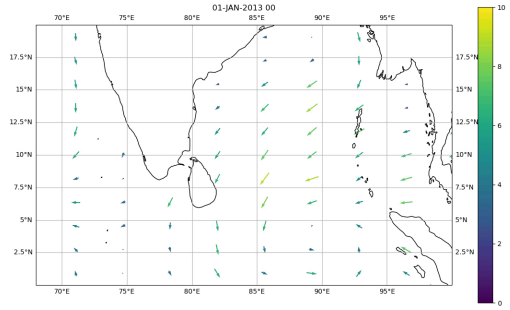

Fig. 35: Quiver Plot using ”interp” sampling.

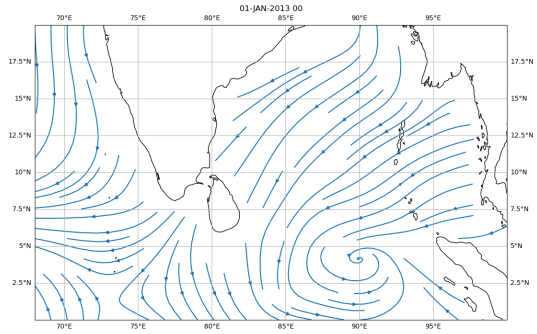

Fig. 36: Streamlines generated for same data.

- $1^{st} March$
- $11^{th} March$
- $21^{st} March$

The preprocessing of the data is done via terminating the data lines by calculating the minimum of the maximum of all the available data points in each latitude. Each of the latitude lines is terminated at this calculated value.

Implementation: Cartopy and Matplotlib are used for plotting the contour plots. As only the coastlines of different countries are important, Plate Caree projection from Cartopy is used. The points at which data is not available i.e. are on land have values -999. The Cartopy coastline functions makes it easy to ignore these values. There are some values which are marked as $-1 * 10^3 4$ which are replaced with $-1 * 10^1 0$, which is ignored.

The contour values from -1 to 32 with a step of 4 are chosen to be plot on the Contour Plot. The colour map viridis is chosen as it goes from Yellow signifying hotter temperatures and Blue signifying colder temperatures. The dpi of 600 is chosen so that when the image is zoomed in on, the details are clearly visible. Python Image Library is used to create a gif of the rendered and saved plots.

For the plot, scan line fill is used as the exact regions do not matter in big distances like in a map. For example, the degree of precision in terms of Latitude is about 30 kilometers. Due

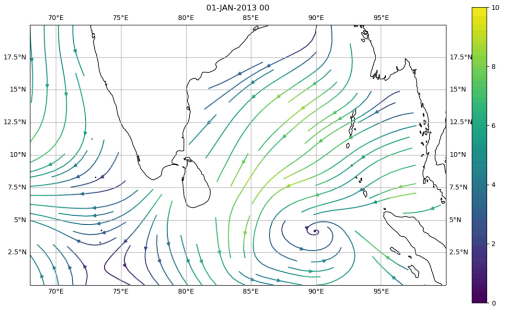

Fig. 37: Streamlines with colormap.

to this reason the exact contour line is not required on the map.

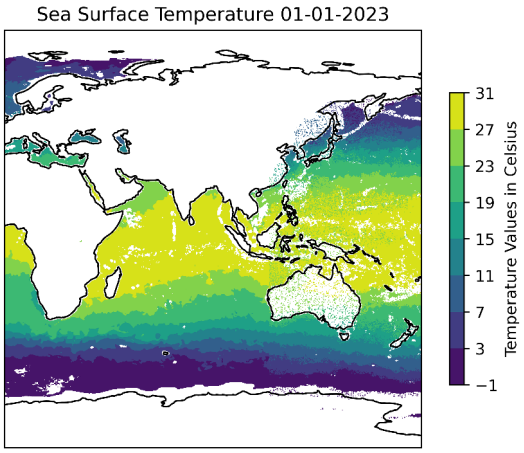

Fig. 38: Plot for 01-01-2023

Inferences: Figures 38, 39 and 40 show plots from 1^{st} January, 1^{st} February and 1^{st} March respectively. The contours can be seen shifting slightly as time progresses. This effect is more apparent when looked at the animation created using all the plots.

As expected, the region near the equator has the highest temperature and it decreases gradually as we approach either poles. The middle band of temperature ranges from 23-31 °C can also be seen to widen a bit from January to March as the Earth passes the Vernal Equinox.

Color Map

Dataset: The AMSR2 ocean dataset from Incois here was used to plot color maps. I used the Sea Surface Temperature dataset. The 8 dates from the first 3 months of 2023 were chosen to plot. The dates are as follows:

- $1^{st} January$
- $11^{th} January$
- $21^{st} January$
- $1^{st} February$
- $11^{th} February$

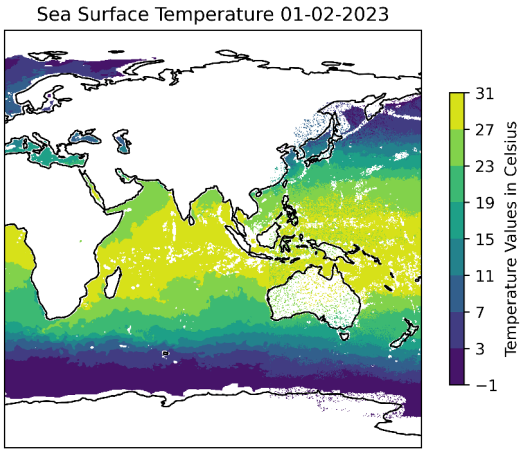

Fig. 39: Plot for 01-02-2023

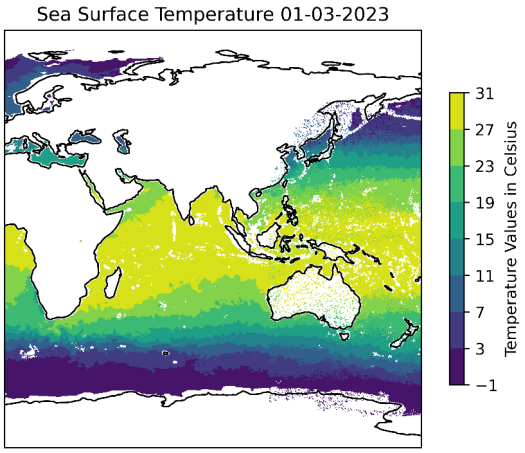

Fig. 40: Plot for 01-03-2023

- $21^{st} February$
- $1^{st} March$
- $11^{th} March$

The dataset contains gridded data in .txt files. They are encoded in ASCII format and include metadata and gridded values for sea surface temperatures. There are also some special values, such as a bad flag, indicating missing or unreliable data. Files are organized in tabular format with latitude, longitude, and then SSTs at different locations.

1) Preprocessing: The data preprocessing involves concluding each data line by determining the minimum of the maximum values across all available data points within each latitude. The termination point for each latitude line is determined based on this calculated value. For preprocessing, I have used the same code that was used to do preprocessing for Contour maps as we use the same dataset for both plots.

2) Implementation: The code loops through the sea surface temperature data, creating a tricontourf plot on a map, and saving each plot as a PNG file. The colormap represents different temperature values, and the code takes care of handling outliers and setting up the plot environment.

The color map is defined with a range of values from -1 to 32 in steps of 0.05. Then I experimented with different color maps ie, diverging, sequential, and also continuous and discrete maps.

The main difference between sequential and diverging colormaps lies in how they represent the progression of data values. Sequential colormaps are designed for representing data where values vary continuously from low to high or high to low. They are suitable for datasets without a clear midpoint or where the emphasis is on the magnitude of values. Diverging colormaps are designed for datasets with a clear midpoint or critical value. They are suitable for emphasizing deviations from a central value, making it easy to identify positive and negative extremes.

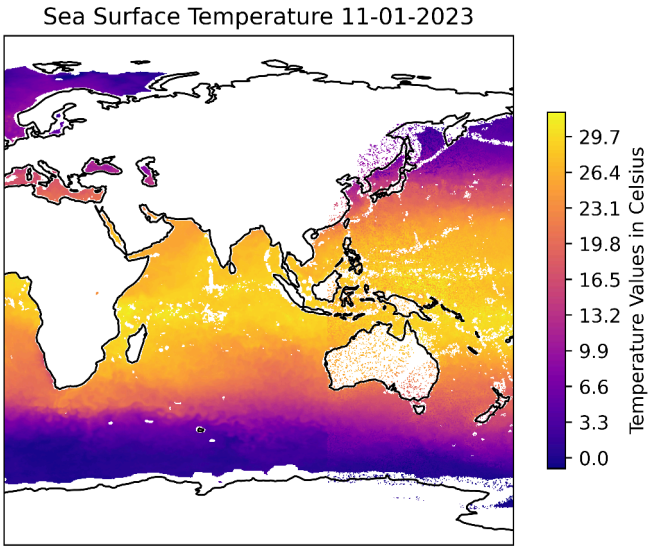

Fig. 41: SST 11-01-2023

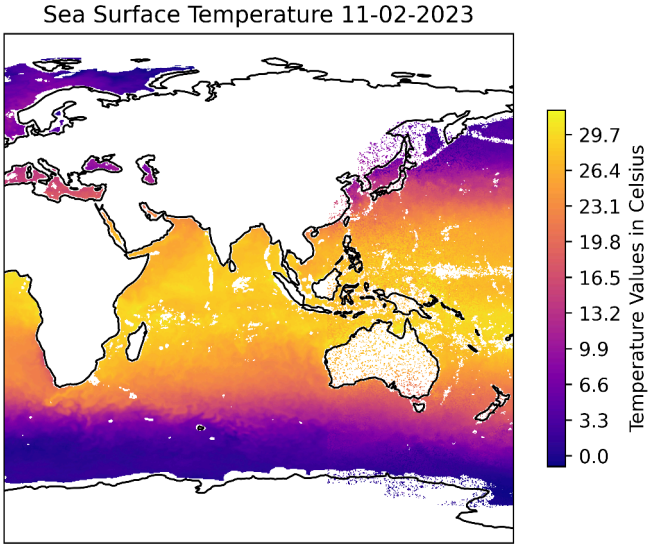

Fig. 42: SST 11-02-2023

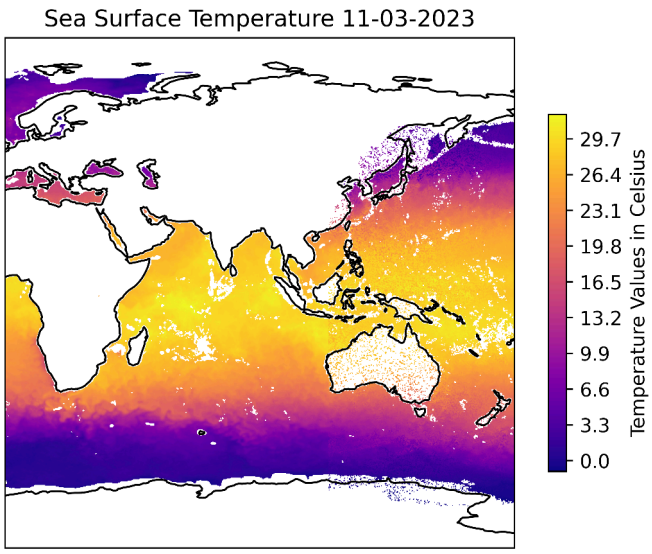

Fig. 43: SST 11-03-2023

3) Plots with inferences: Figure 41,42,43 are sea surface temperature color plots of 3 days from the first 3 months of 2023. Color gradients display a slight variation over time, particularly evident in the gif file uploaded. Also, the region near the equator showcases the warmest colors, gradually transitioning to cooler hues towards the poles. These observations are influenced by the chosen color maps, contributing to the visual representation of temperature gradients and changes.

This is clearly seen when you plot the same using a diverging color plot. Figure 44 is an example of a diverging plot where you can have lighter shades denoting cooler temperatures and warmer shades denoting higher temperatures.

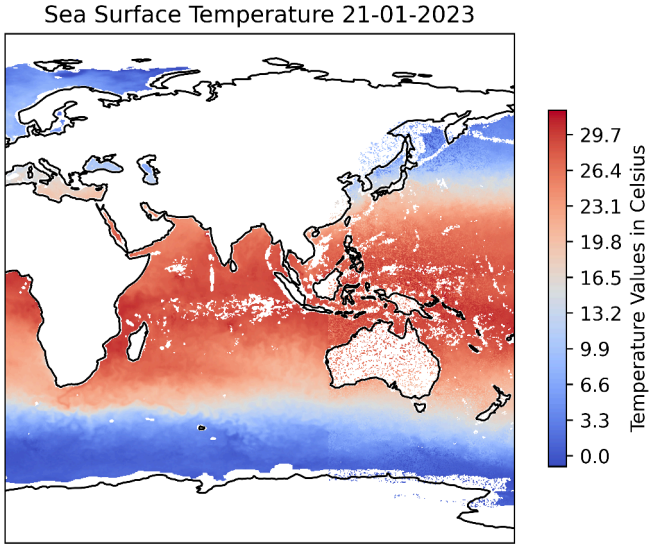

Fig. 44: SST 21-01-2023

If we were comparing graphs across all the months in a year, we would be able to see changes in sea surface temperature

over different seasons.

plasma colormap which is a sequential colormap was used for the first three diagrams and **coolwarm** which is a diverging colormap was used for the next diagram.

I have also tried using the discrete color map in **tab10** but that makes the diagram difficult to analyze (Fig 45).

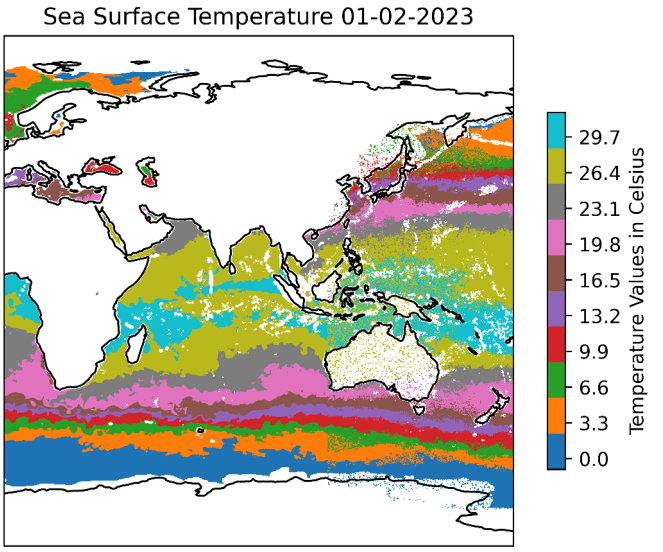

Fig. 45: SST 1-02-2023

REFERENCES

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Fast unfolding of communities in large networks, in Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000

[2] Jacomy M, Venturini T, Heymann S, Bastian M. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. PLoS One. 2014 Jun 10;9(6):e98679. doi: 10.1371/journal.pone.0098679. PMID: 24914678; PMCID: PMC4051631.

[3] Martin, S., Brown, W. M., Klavans, R., and Boyack, K. W., "OpenOrd: an open-source toolbox for large graph layout", in Visualization and Data Analysis 2011, 2011, vol. 7868. doi:10.1117/12.871402.