Data Visualization: Assignment 2

Vidhish Trivedi
IMT2021055
IIIT - Bangalore
Bangalore, India
Vidhish.Trivedi@iiitb.ac.in

Sai Madhavan G
IMT2021101
IIIT - Bangalore
Bangalore, India
g.saimadhavan@iiitb.ac.in

Krutik Patel
IMT2021024
IIIT - Bangalore
Bangalore, India
Krutik.Patel@iiitb.ac.in

I. Scientific Visualization

A. Scalar Field Visualization - Color Mapping

Color mapping, also known as colormap or color scale, is a technique used in scalar field visualization to represent variations in a scalar quantity across a spatial domain. In scalar field visualization, we typically have a scalar field, which is a function that assigns a scalar value to each point in space. This scalar value could represent physical quantities such as temperature, pressure, density, or other scalar properties.
Color mapping involves associating colors with specific scalar values to visually convey the variations in the scalar field. This helps in understanding the distribution and patterns of the scalar quantity within the given domain. The choice of colors and their arrangement is crucial in accurately conveying information about the scalar field.

1) Dataset - Color Mapping: We use the AMSR2 Ocean Dataset [1] to visualize *surface rain rate*. The dataset consists of the daily data, sampled in a 3-day wise methodology for a period of 10 years. We choose a contiguous period of three months starting from May, 2013 till July, 2013. Ten dates are sampled from this period, each being 8 to 10 days apart from others. The team decided on this period, as it covers a significant portion of monsoon for the Indian subcontinent. The choice of year was made arbitrarily.

2) Data Processing: The data for specific dates was downloaded in the netCDF format, and was processed using the netCDF4 [2] library in python. The *surface rain rate variable* was extracted, giving us the scalar values corresponding to 1440 longitudes and 720 latitudes. Furthermore, the bad values (indicated by -999) were replaced with NAN using the numpy [3] library.

3) Implementation: To transform and visualize the data, we employed the pyplot [4] API layer in in Matplotlib. Our implementation includes experimentation with different color palettes and color scales. Through multiple visualizations and experimentation, we propose that for our particular usecase, a discrete color map based on Viridis color palette is appropriate.

4) Experiments On Color Mapping Palettes: Various color palettes were compared, such as PiYG, PrGN, BWR, which are diverging color palettes, and Hot, Magma, Viridis, which are sequential color palettes.

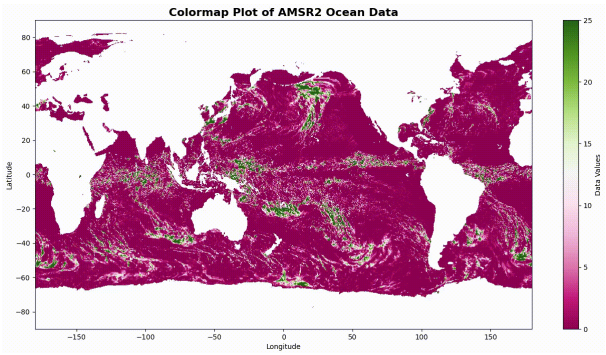

Fig. 1. Divergent PiYG Palette.

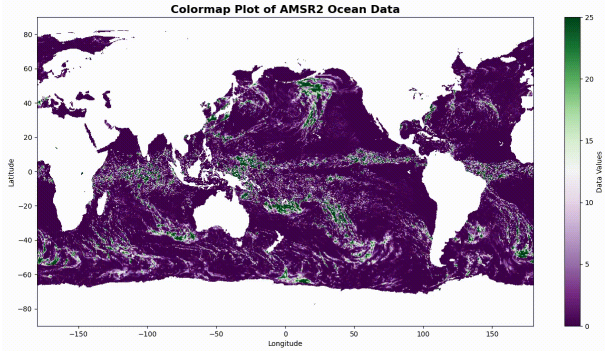

Fig. 2. Divergent PrGN Palette.

Based on these experiments, we make the following observations:

- Any color palette which maps a scalar value to white or a near-white color would be unsuitable, since the bad values (-999) and land is also indicated through the use of white color. This implication is apparent in Fig 1, Fig 2, Fig 3, Fig 4, and Fig 5. Such color palettes are likely to introduce artefacts in the visualizations and should be avoided.
- We also observe that the majority of scalar values in our dataset are greater than the center of the range of values,

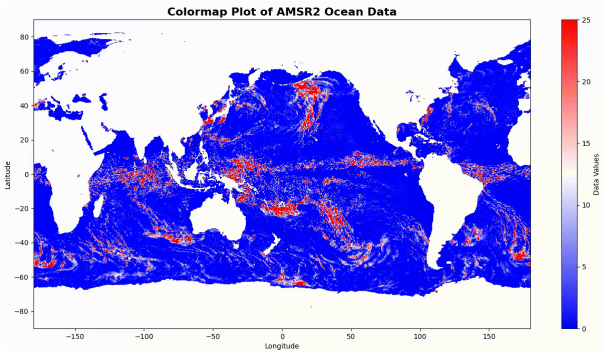

Fig. 3. Divergent BWR Palette.

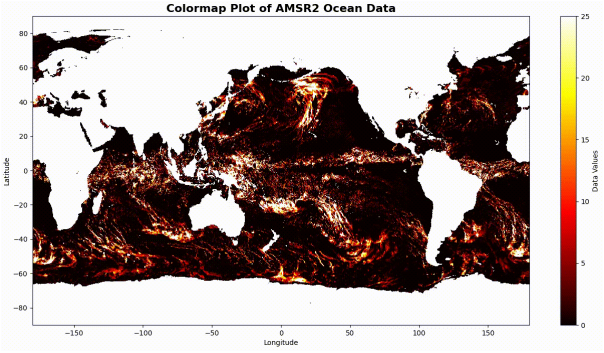

Fig. 4. Sequential Hot Palette.

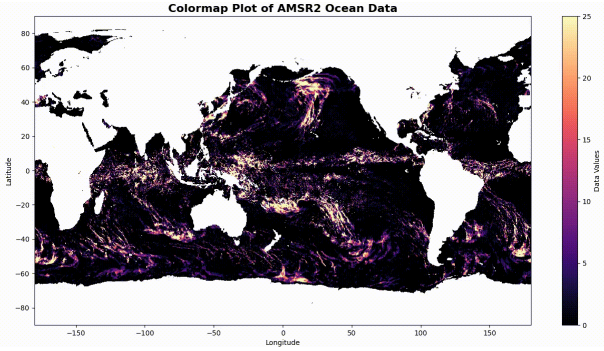

Fig. 5. Sequential Magma Palette.

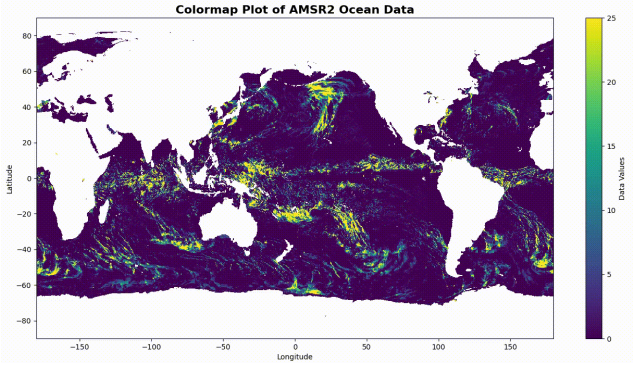

Fig. 6. Sequential Viridis Palette.

which is roughly 12.5. Using a diverging color palette would have made sense if the data extended to both sides of such a central value, which is clearly not the case here. Moreover, we could choose some other value to diverge the palette from but doing so would make the visualizations harder to understand and interpret across timesteps.

These observations led to the choice of proceeding with a sequential color palette which does not introduce artefacts. For our purpose, we chose to proceed with the *sequential Viridis color palette*, which has the added benefit of being a *perceptually ordered color map*. Henceforth, we shall consider only this color map for further experiments and visualizations.

5) Experiments On Parametric Mapping: Primarily, there are two types of parametric color mapping: using global maxima and minima, and using local maxima and minima.

- **Global Mapping** is suitable when one wants a consistent color scheme for the entire dataset, making it easier to observe long-term trends and identify outliers across all timesteps. It is useful when we want to emphasize the overall distribution of scalar values. This approach ensures that color representations remain stable and consistent, allowing viewers to compare different timesteps directly based on the global scalar value range.
- **Local Mapping** is beneficial when you want to focus on temporal changes or localized features in the data. It provides a more granular view of each timestep, which can be helpful in tracking specific events or phenomena as they evolve over time. Using local maxima and minima can help highlight temporal variations and anomalies in the scalar field over time, but it may make direct comparisons between timesteps more challenging.

Based on this comparison and given that we wish to observe and interpret trends across multiple timesteps, we chose to proceed with global mapping, i.e. using global maxima and minima. The subset of data which we sampled has a global minima of 0 (barring bad values) and a global maxima of 25.0. In any case, for the chosen subset of data, the global maxima and minima do not vary significantly compared to the local maxima and minima.

6) Experiments On Color Mapping Scales: Three types of scales, namely Continuous, Discrete, and Logarithmic, were considered for mapping scalar field values to colors.

- **Logarithmic Scale:** As the logarithm of 0 (the global minima) approaches negative infinity, we consider 10^{-4} as the minimum when using a logarithmic mapping scale. The *clip* parameter is set to *True* while creating the normalization object, ensuring that values below 10^{-4} are also mapped to the lowest color on the color map. Logarithmic scales are useful when the scalar values are very large.
  Since we have established that in our subset of data, the global maxima and the global minima are 25 and

0 respectively, there is no apparent need of using a logarithmic scale.
Moreover, using such a scale also leads to the colors ”blending” into one another. This is primarily due to the scalar values lying in a very small range (0 - 25). Hence, it is difficult to interpret and make inferences in such a visualization, as is evident in Fig 7.
The visualization fails to highlight significant changes in the value of *sea surface rain rate*, and at the same time, visually amplifies the presence of smaller scalar values, thereby contaminating any inferences that would have been possible.

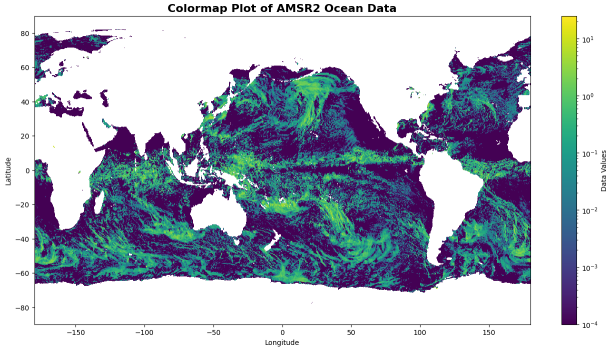

Fig. 7. Logarithmic Scale with Viridis Palette.

- **Continuous Scale:** In order to use a continuous scale, we first normalize the data, originally in the range (0 - 25). When the scalar field was visualized without normalization, no discernible differences were observed. Thus, we decided to visualize the field without normalization, as can be seen in Fig 8. Note that to make the visualization easy to interpret, the color bar is representative of the actual scalar values of rain rate, and not the normalized values.
  The continuous scale provides us with relatively better results compared to the logarithmic scale. The smaller scalar values are mapped to an appropriate color on the scale, which is indicative of their true value in the range (0 - 25). Hence, the plot is no longer contaminated with false representations of scalar values.

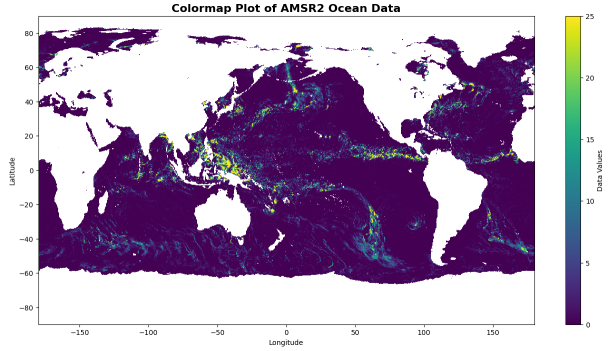

Fig. 8. Continuous Scale with Viridis Palette.

- **Discrete Scale:** We first linearly scale the values using the Normalize class in matplotlib.colors, and then introduce discrete color bands of length 0.2 each using the BoundaryNorm class. These are mapped to specific colors in the Viridis color palette. We also use a special color, red, to indicate values values which are notably high (more than 2.0 on the new scale).
  We find that the discrete scale is the best-suited color mapping scale out of the three, as it does not introduce any artefacts, nor does it contaminate the visualization. Moreover, it allows us to specify discrete bounds and their colors, using which we can focus on a particular range of values (in this case, we focus on values which are higher than 2.0 on the new scale) while also accurately representing other values. This is shown in Fig 9.

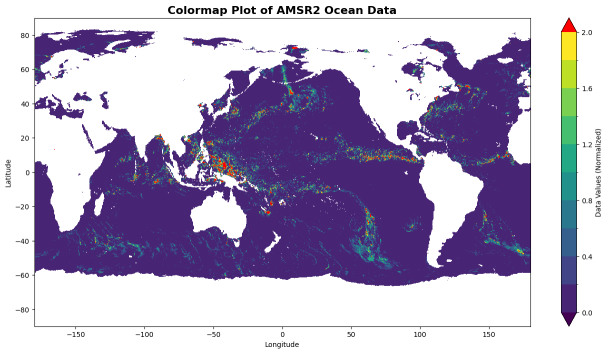

Fig. 9. Discrete Scale with Viridis Palette.

Based on the above experiments and observations, we choose the discrete color mapping scale as the best-suited scale for our purposes. Animated GIF files for the three experiments pertaining to color mapping scales have been attached with the submission, they can be found at ”project_root /GIF/viridis_{scaling_strategy} ”.

7) Observations and Inferences: We start with day 01, May 02, 2013 and visualize the scalar field values representing sea surface rain rate for 10 days in 2013, which are: May 02, May 12, May 22, June 02, June 12, June 22, July 02, July 12, July 20, and July 30. The selection of dates was done with the idea that monsoon in India spans the months June - August roughly. The inferences are verified using [15] and via Google Search (for timeline of monsoon in India).

- We begin in the month of May, which falls in the Indian summer season. We can see that the value of sea surface rain rate is low around the Indian coastline. This is expected given that monsoon is yet to arrive (Day 01 - 03). This can be observed in Fig 10, Fig 11, and Fig 12.
- Proceeding forward, we see that the rain rate increases as we enter June, and significantly higher values are recorded in mid June in the Arabian Sea along India’s coastline. This is also in accordance to the fact that the South-East Monsoon arrives at the Indian mainland at Kerala, usually by the first week of June. This is evident

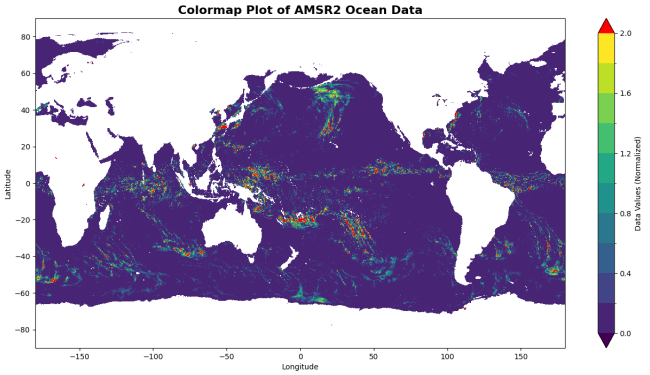

Fig. 10. Day - 01, May 02, 2013

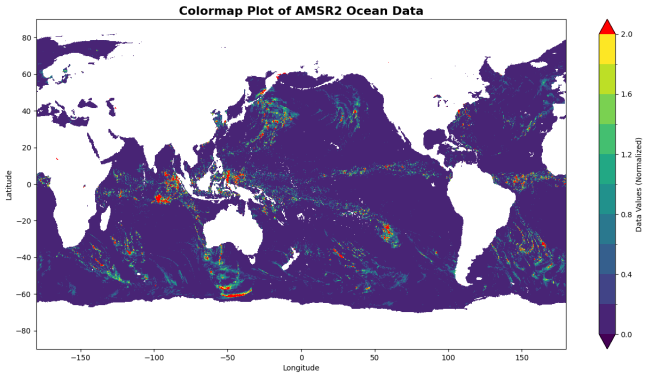

Fig. 11. Day - 02, May 12, 2013

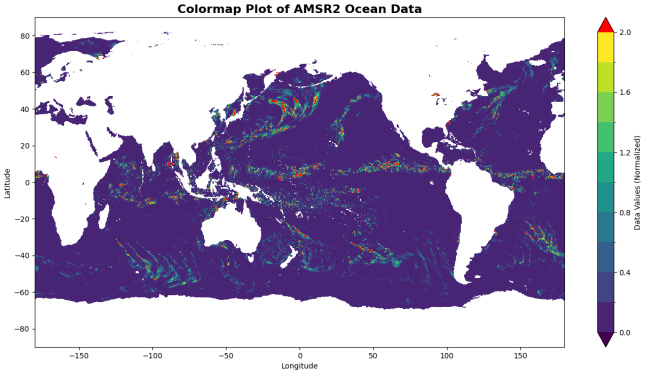

Fig. 12. Day - 03, May 22, 2013

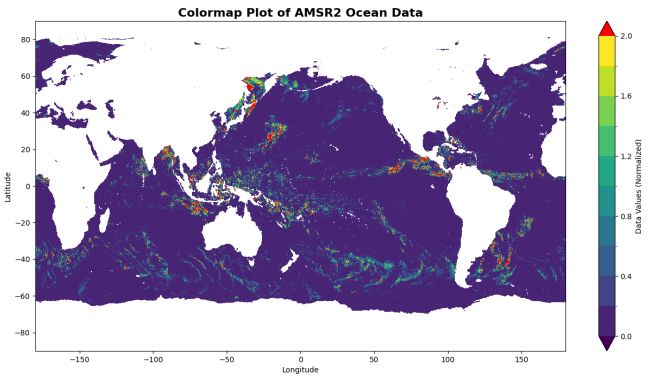

Fig. 13. Day - 04, June 02, 2013

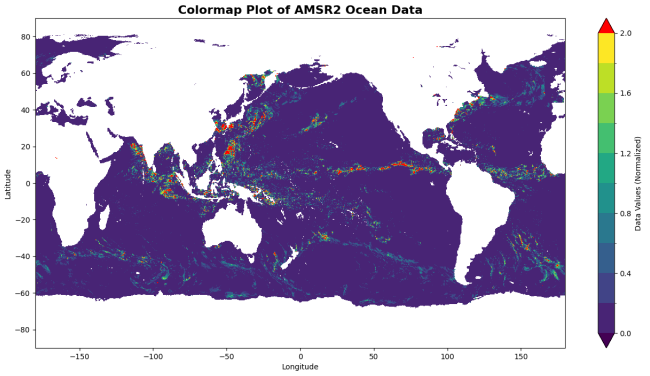

Fig. 14. Day - 05, June 12, 2013

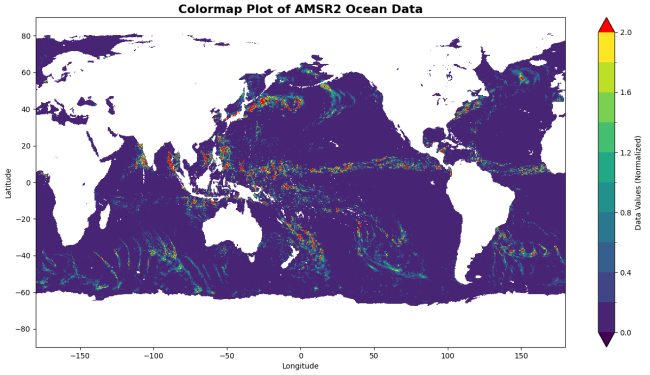

Fig. 15. Day - 06, June 22, 2013

in Fig 13, Fig 14, and Fig 15 (Day 04 - 06). This behaviour of rain rate persists till the end of June.

- As we enter July, we observe that the monsoon on the western coast of India starts to recede, and the rain rate drops in the Arabian Sea, except near Kerala, where the receding monsoon maintains a relatively higher rain rate, see Fig 16 (Day 07).
- By mid July, the South-East Monsoon crosses India and Bay of Bengal experiences a rise in sea surface rain rate, see Fig 17 and Fig 18 (Day 08 - 09).
- As the month of July ends, we see that the rain rate values drop in the Bay of Bengal as well, indicating that the monsoon has moved further north-east towards Tibet and China, see Fig 19 (Day 10).

B. Vector Field Visualization - Quiver Plotting

1) Data Selection: We selected a sample of 10 days between December 2011 and January 2012 uniformly from the OSCAT wind data set [1]. This specific time frame allows us to visualize the impact of Cyclone Thane, which struck

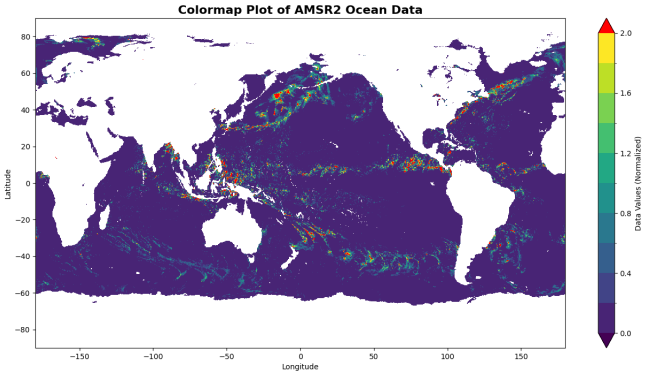

Fig. 16. Day - 07, July 02, 2013

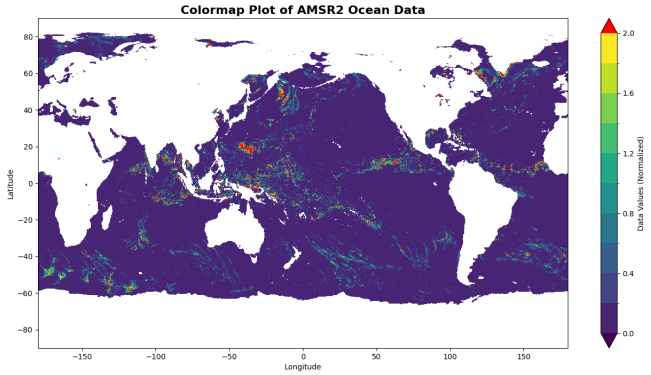

Fig. 17. Day - 08, July 12, 2013

South India in the Bay of Bengal between December 25 and December 31, 2011.

2) Implementation: To transform and visualize the data, we employed the Basemap [5] library in Matplotlib. Our implementation includes two variations: one with vector lengths varying based on magnitude and another where vector lengths are uniform, and magnitude is represented using color as a channel.

3) Experiments:

- **Selecting the Region:** Initially, attempting a quiver plot on the world map provided limited insights due to the necessity of a sparse grid space for meaningful vectors. Therefore, we focused our plot exclusively on the Bay of Bengal (Fig 20) to observe how Cyclone Thane affected wind patterns during that period.

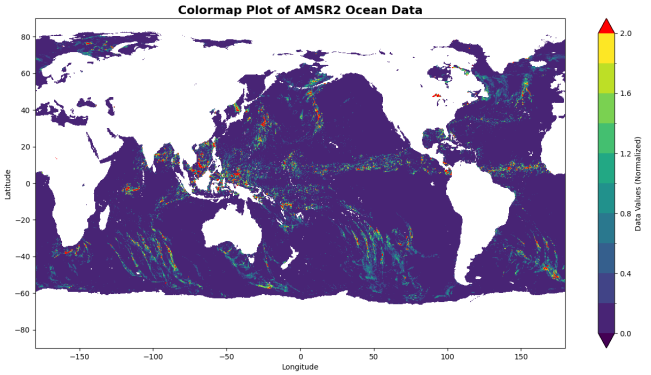

Fig. 18. Day - 09, July 20, 2013

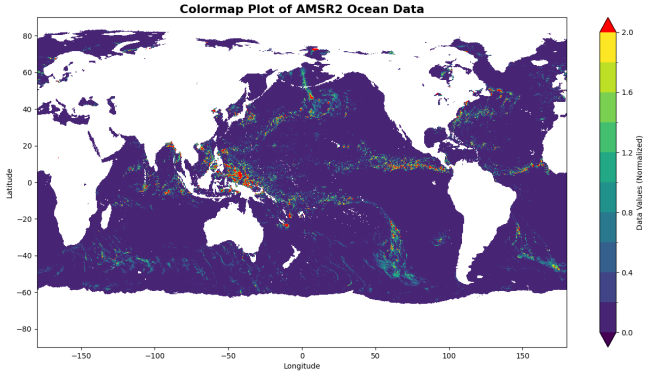

Fig. 19. Day - 10, July 30, 2013

Fig. 20. The selected study region centered on the Bay of Bengal.

- **Selecting the Number of Grid Points:** Once our study region was defined, we needed to resample grid points for plotting the quivers. Experimenting with different numbers (Fig 21, 22, 23), we found that a 21x21 point grid struck a balance between localized representation and vector visibility.
- **Selecting Colormap:** In our experimentation with unit vectors and magnitude represented by color, we explored different colormaps. Considering the sequential nature of the data, we deliberated between a perceptually uniform sequential colormap like 'viridis' (Fig 24) and a monochromatic sequential colormap like 'Blues' (Fig 25).

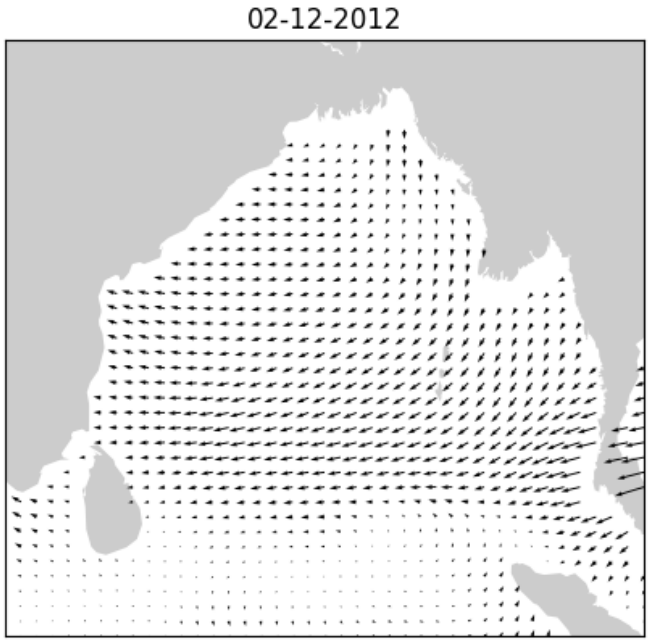

Fig. 21. Resampling with 41x41 points. Although vector directions are visible, determining magnitudes is challenging

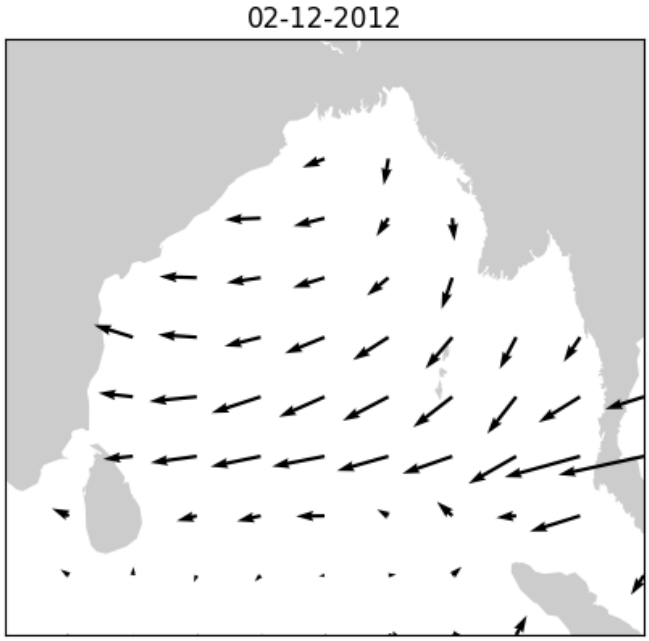

Fig. 22. Resampling with 11x11 points. Magnitude differences are apparent, but points are too spread out for meaningful inference.

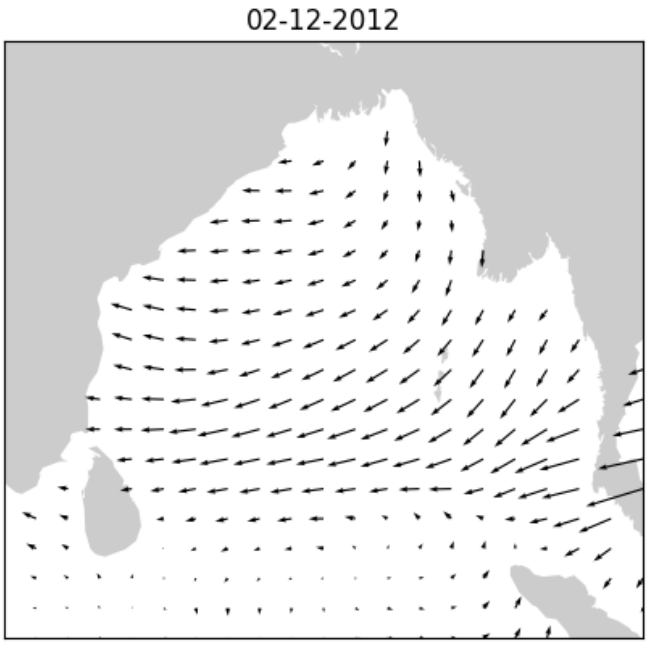

Fig. 23. Resampling with 21x21 points. Densely placed vectors without sacrificing individual vector size.

We opted for 'viridis' as it represented differences in values better than 'Blues.'

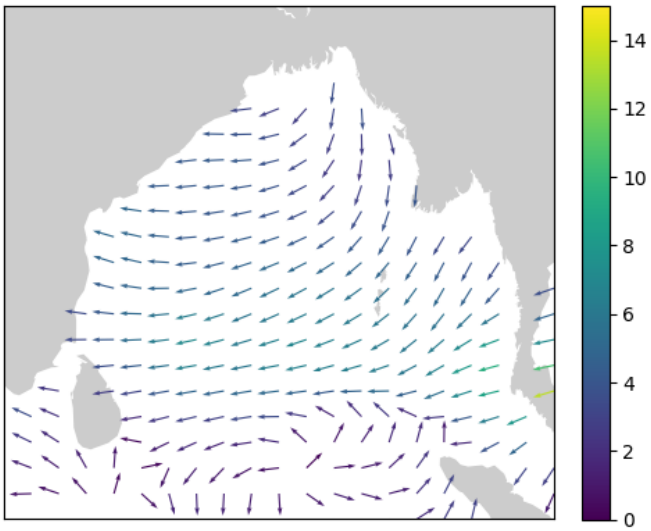

Fig. 24. Plot generated using 'viridis' colormap.

4) Plots Generated:

- **Quiver Plots:** Fig 26-35
- **Quiver Plots with Colormap:** Fig 36-45

5) Observations: Our observations include:

- Wind speeds predominantly exhibit a south-west direction, characteristic of the north-east monsoon over the Bay of Bengal during this period.

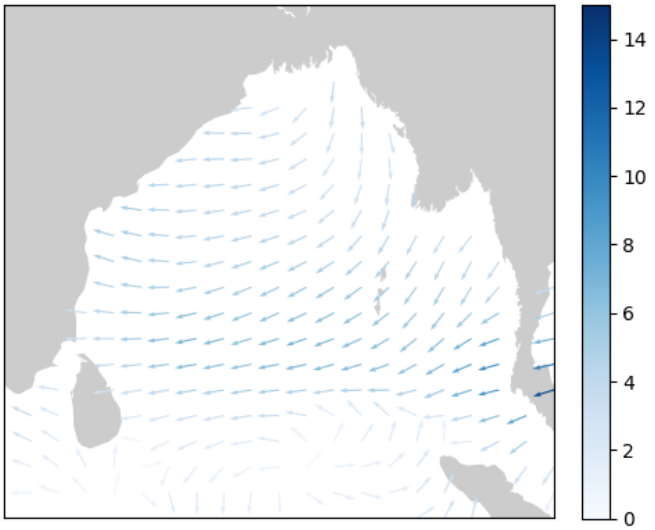

Fig. 25. Plot generated using 'Blues' colormap

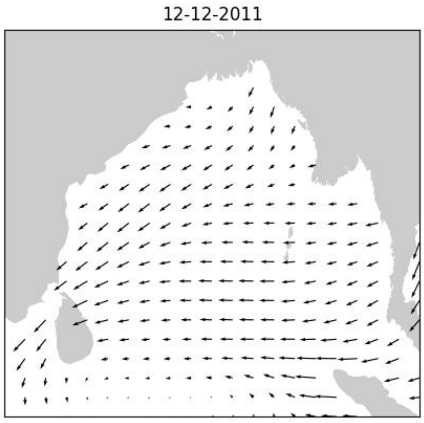

Fig. 26. Quiver plot generated for 12-12-2011.

- The formation of Cyclone Thane is evident as a counter-clockwise swirl in the southern part of the map on dates 21-12-2011 (Fig 29 & 39) and 24-12-2011 (Fig 30 & 40).
- On 27-12-2011 (Fig 31 & 41), the cyclone is fully formed and has moved north, with the maximum velocity observed around its periphery.
- By 30-12-2011 (Fig 32 & 42), the cyclone makes landfall near the coasts of Tamil Nadu and Andhra Pradesh.
- Subsequent days reveal minor disturbances in the southern part of the map.

In conclusion, these observations provide valuable insights into the wind patterns and the progression of Cyclone Thane during the specified period.

C. Scalar Field Visualization - Contour Plotting

1) Data Selection: We selected a sample of 3 days between May 2013 and July 2013 uniformly from the OSCAT wind

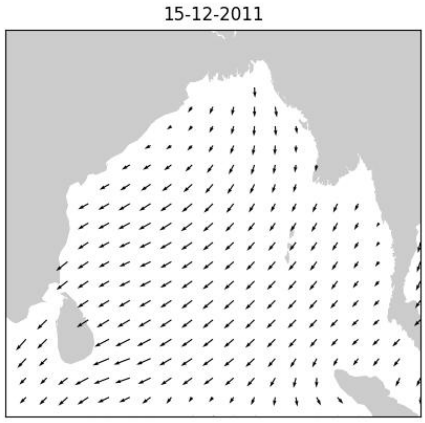

Fig. 27. Quiver plot generated for 15-12-2011.

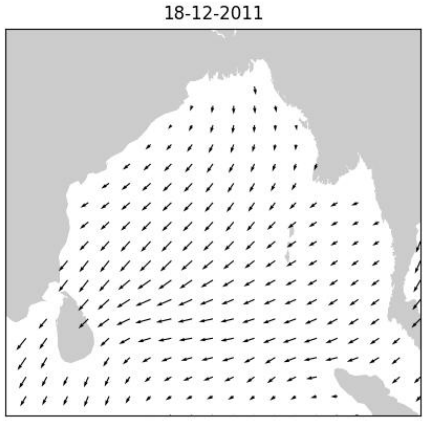

Fig. 28. Quiver plot generated for 18-12-2011.

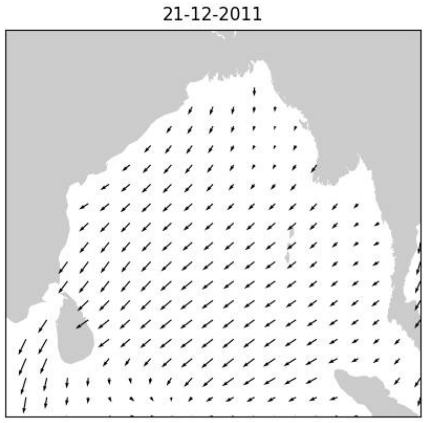

Fig. 29. Quiver plot generated for 21-12-2011.

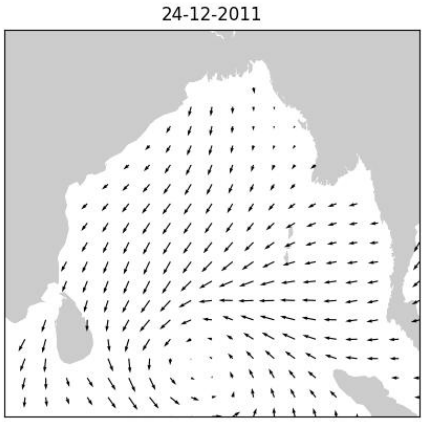

Fig. 30. Quiver plot generated for 24-12-2011.

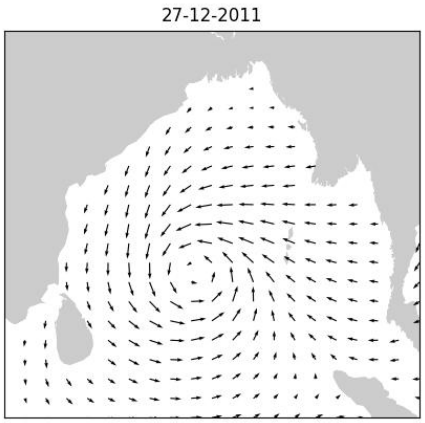

Fig. 31. Quiver plot generated for 27-12-2011.

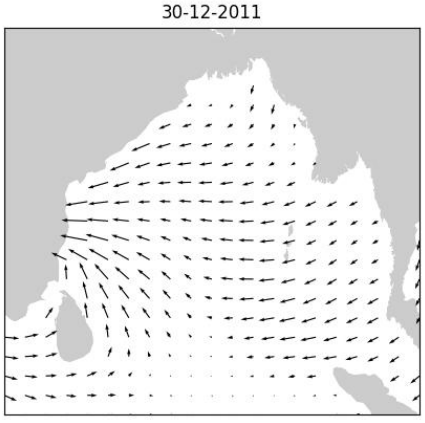

Fig. 32. Quiver plot generated for 30-12-2011.

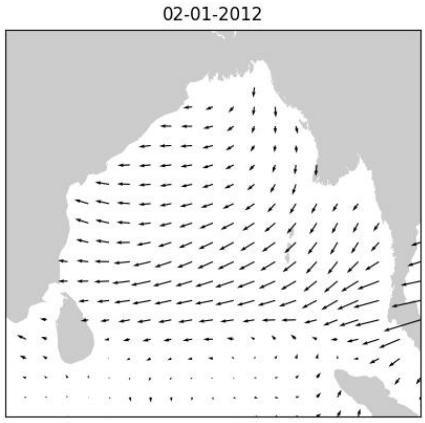

Fig. 33. Quiver plot generated for 02-01-2012.

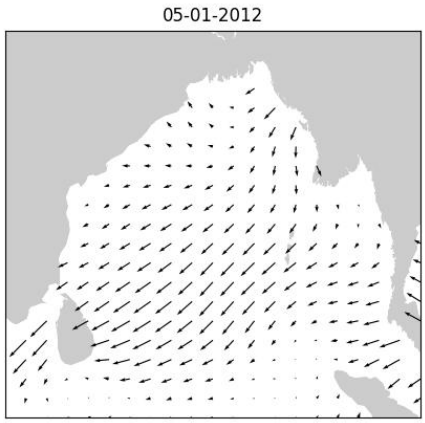

Fig. 34. Quiver plot generated for 05-01-2012.

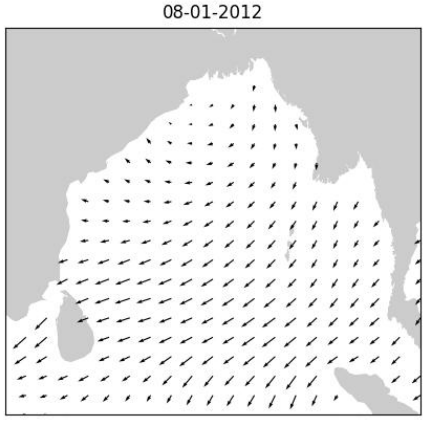

Fig. 35. Quiver plot generated for 08-01-2012.

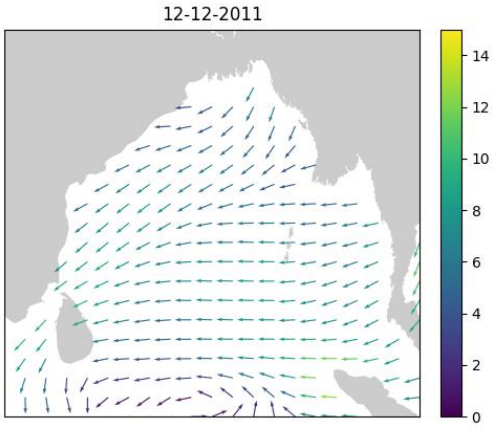

Fig. 36. Quiver plot with colormap generated for 12-12-2011.

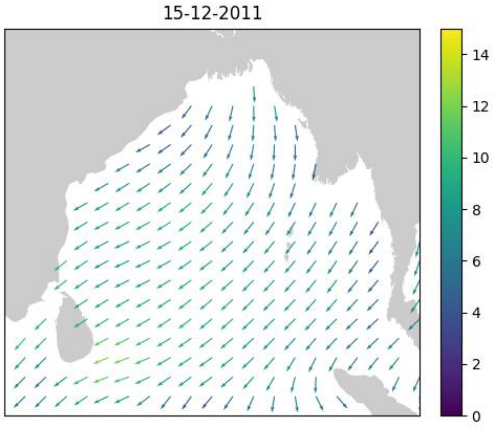

Fig. 37. Quiver plot with colormap generated for 15-12-2011.

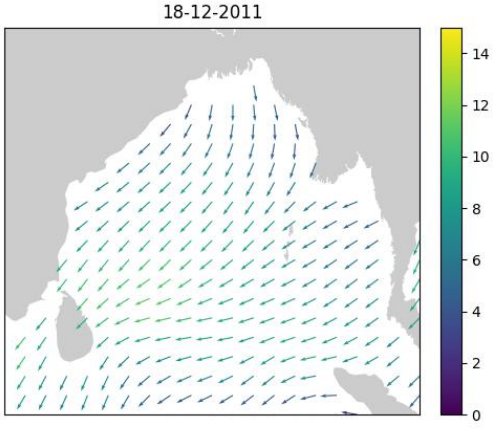

Fig. 38. Quiver plot with colormap generated for 18-12-2011.

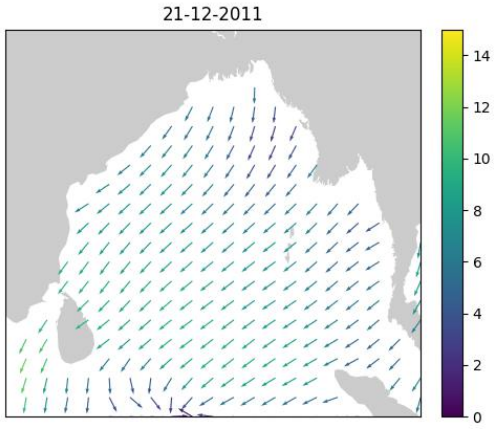

Fig. 39. Quiver plot with colormap generated for 21-12-2011.

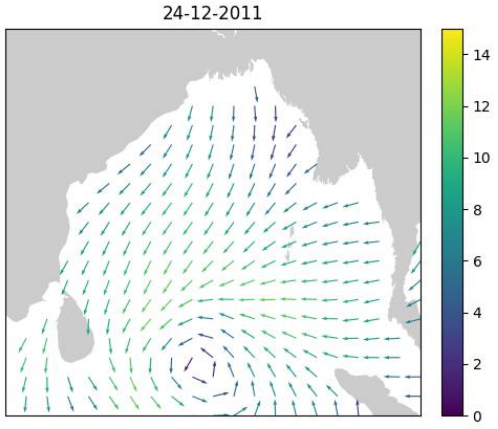

Fig. 40. Quiver plot with colormap generated for 24-12-2011.

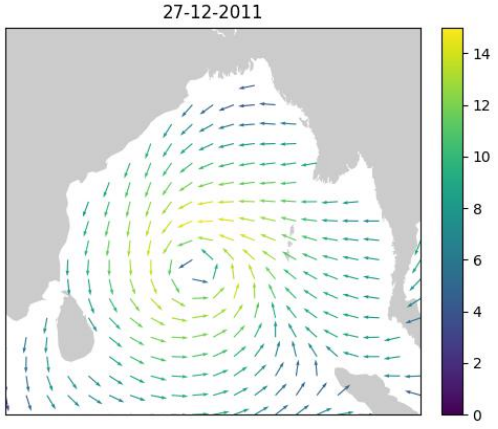

Fig. 41. Quiver plot with colormap generated for 27-12-2011.

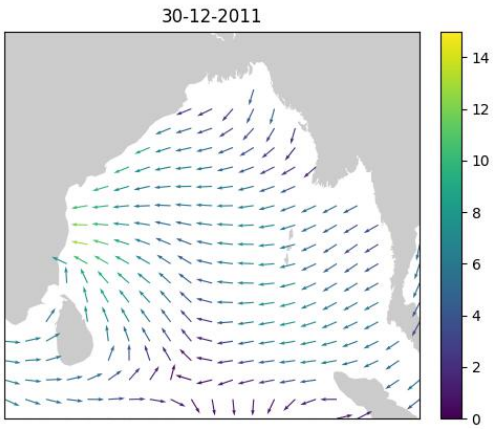

Fig. 42. Quiver plot with colormap generated for 30-12-2011.

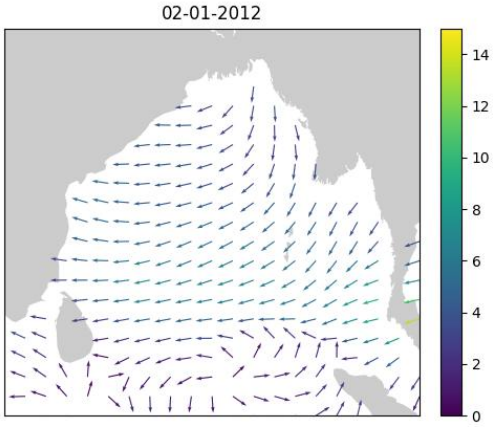

Fig. 43. Quiver plot with colormap generated for 02-01-2012.

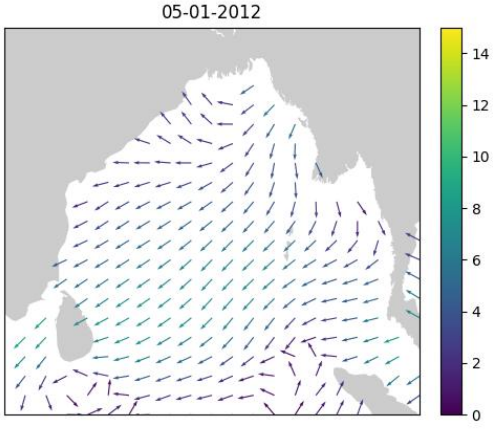

Fig. 44. Quiver plot with colormap generated for 05-01-2012.

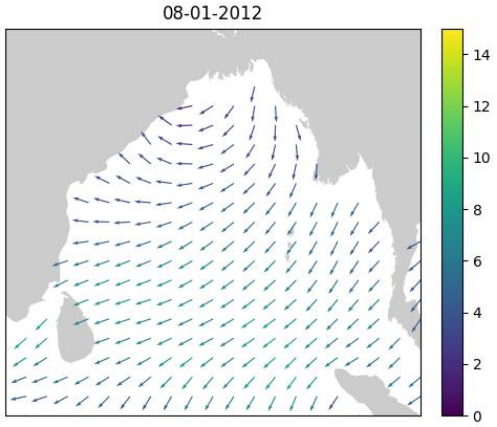

Fig. 45. Quiver plot with colormap generated for 08-01-2012.

data set [1]. This specific time frame allows us to visualize the onset of monsoon in different countries.

2) Implementation: To transform the dataset, we utilized several Python libraries, including xarray [17], netCDF4 [18], numpy [3], and pandas [13]. For image generation, we employed matplotlib [4] and cartopy [20] libraries, and for creating gifs and images, the imageio [19] library was used. The initial step involved converting the dataset from .nc format to a pandas [13] dataframe. Subsequently, the data was reshaped into a dataframe with dimensions (1440, 720). Meshgrids for longitude and latitude were then created and utilized with the reshaped data to generate a contour map. The visualization employed the 'jet' colormap with 20 contour levels.

3) Experiments: :

- We experimented with various datasets and ultimately chose the one that yielded the most effective visualization, specifically focusing on water vapor.
- In our contour plotting efforts, we experimented with two methods: contour fill and marching squares.
- **Marching Squares:** We generated contour plots (Fig. 46, 47, 48, & 49) employing the marching squares algorithm and conducted experiments with different plot configurations, including colormap, contour levels and line widths to get the most visually appealing plot. However, we observed that these visualizations posed challenges for regular readers to comprehend when utilizing the Marching squares. This is also discussed later in this subsection.
- **Contour Fill:** Additionally, we generated contour plots (Fig. 50) using the contour fill method, once more adjusting settings like colormap and contour levels. Through experimentation, we determined that the optimal plot was achieved using the 'jet' colormap and 20 contour levels. Consequently, we observed that the contour fill method produced a better plot compared to the marching squares method. The reasons for this preference are explained in

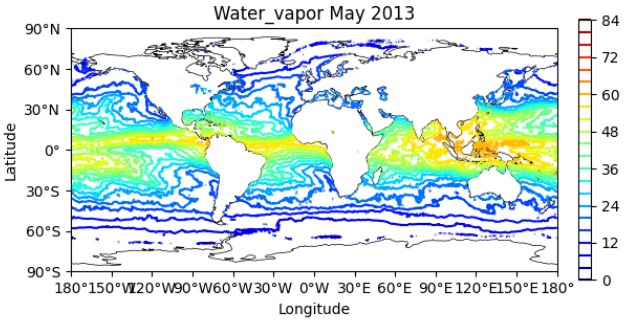

Fig. 46. Marching squares water vapor plot generated with 20 Levels, line width 1 pt., jet colourmap

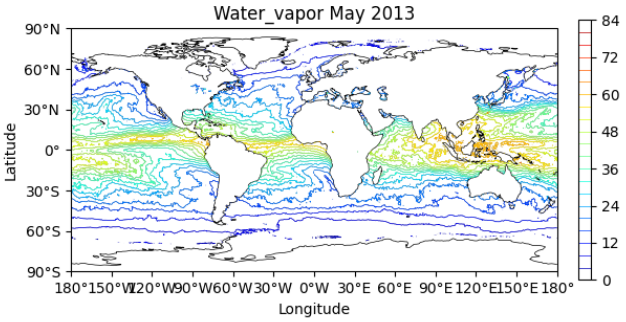

Fig. 47. Marching squares water vapor plot generated with 20 Levels, line width 0.5 pt. 'jet' colourmap

the following section.

ContourFill: This method tends to produce smoother and more visually appealing contour plots. The filled contours provide a continuous gradient of colors, allowing for a clear representation of the gradual changes in water vapor content. The filled regions enhance the overall visual coherence of the plot. Filled contours simplify the interpretation of the plot. The gradual color transitions make it easier to identify trends and patterns in water vapor content, facilitating a more intuitive understanding of the data. Meanwhile the step-like contours for water vapor content produced by marching squares may require additional effort to interpret, like one shown in Fig 46.

The contour fill algorithm was chosen to effectively highlight regional differences in water vapor content. Filled contours quickly show areas with significant changes, like the start

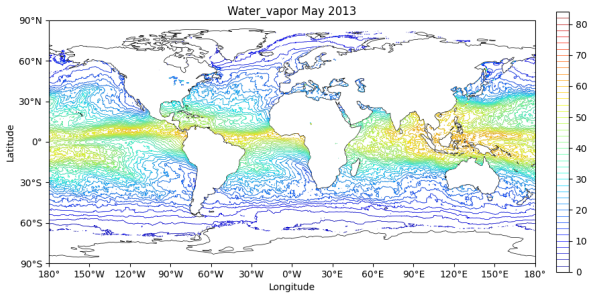

Fig. 48. Marching squares water vapor plot generated with 50 Levels, line width 0.5 pt. 'jet' colourmap

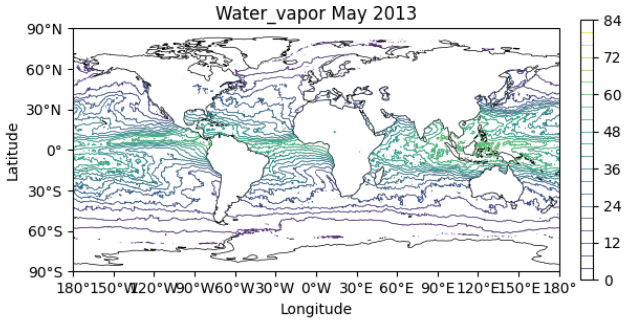

Fig. 49. Marching squares water vapor plot generated with 20 Levels, line width 0.5 pt. 'viridis' colourmap

of the monsoon season. Marching squares did not emphasize regional variations this well. Step-wise contours (marching squares) could hide subtle details in the dataset, just like what we have with us. The blank (white) spaces don't emphasize water vapor content, which is more about regions than lines (step wise contours).Even with different contour levels, interpreting data might be challenging, for many readers and hence some findings may not be well interpreted (may even be missed) in the marching squares contour. Also, the smoother and more attractive contours from the contourfill algorithm can improve the report's visual appeal, while the jagged contours

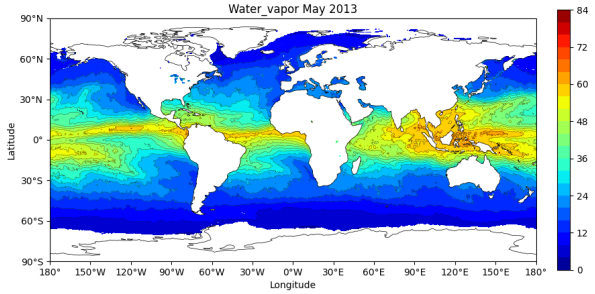

Fig. 50. Contour fill water vapor plot generated with 20 Levels, line width 0.5 pt. 'jet' colourmap

from marching squares may be less visually pleasing and distract from key findings in the water vapor analysis.

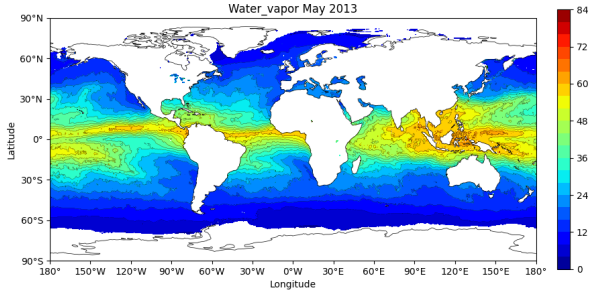

Fig. 51. Contour fill water vapor plot on May 2013

4) Observations and Inferences: : The contour plot presented below illustrates the dynamic changes in water vapor content across different regions and months, offering valuable insights into seasonal variations. Though the map is a world

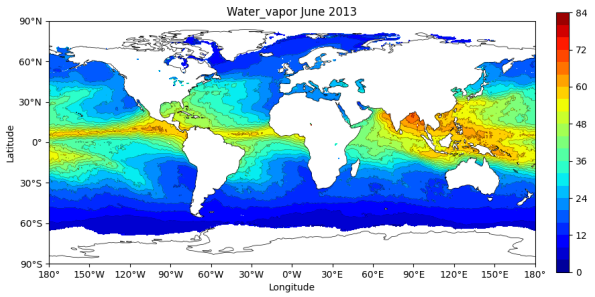

Fig. 52. Contour fill water vapor plot on June 2013

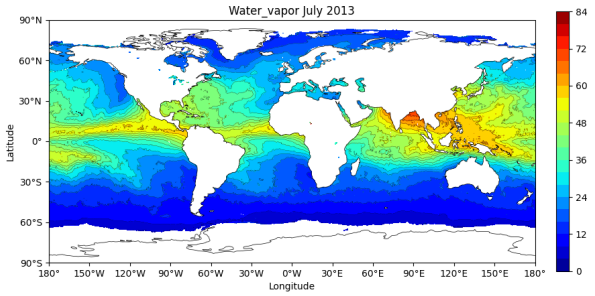

Fig. 53. Contour fill water vapor plot on July 2013

map, the focus lies on the Bay of Bengal, a region known for its climatic significance.

Upon observing the contour plot, it becomes evident that the Bay of Bengal undergoes substantial changes in water vapor content. In May, the region exhibits minimal water vapor, depicted by cooler tones on the plot. This low moisture content may indicate a pre-monsoon period characterized by dry conditions.

However, as we progress to July, a noticeable shift occurs. The Bay of Bengal transforms into vibrant red hues on the contour plot, suggesting a significant increase in water vapor. This change aligns with the typical onset of the monsoon season in the northern states of the Indian subcontinent. The intensification of red tones indicates a surge in atmospheric moisture, a crucial precursor to the monsoons.

Let's now analyze the variations in water vapor near the equators, where the contour plot provides a detailed examination of atmospheric dynamics. By examining the depicted features, we can observe a pattern between solar proximity and the patterns of seasonal monsoons in these regions.

The plot underscores that regions above and near the equator consistently exhibit higher water vapor content. This observation aligns with the principle that the northern hemisphere, being closer to the sun during the initial months of the year, experiences heightened moisture levels. The warmer temperatures in this hemisphere contribute to increased evaporation and atmospheric water vapor.

Additionally, the contour plot reveals the last stages of retrieval of monsoon in the southern hemisphere. Typically experiencing its monsoon season in December, the southern hemisphere displays a distinctive pattern. In May, a subtle light blue shade signifies relatively lower moisture content. However, as the months progress, this hue gradually vanishes.

II. Information Visualization

A. Node-Link Diagrams

1) Dataset: We visualized the "David Copperfield" network from konect.cc [6]. The network consisted of common noun and adjective adjacencies present in the novel "David Copperfield" [7] by Charles Dickens. Each node of the graph corresponds to a word (a noun or adjective) and an edge between two nodes represents the two words occur together. The graph is unweighted and undirected. There are 112 nodes and 425 edges connecting them. Of the 112 words, 59 are nouns and 53 are adjectives.

2) Implementaion: We used gephi [8] for visualizing the graphs using different layouts. We used a Part-Of-Speech tagger from the NLTK [9] library in python to tag each word as either a noun or adjective to improve our analysis.

3) The Fruchterman Reingold Algorithm:

Theory [10]

1) Initialization:
   - Nodes are randomly placed in the drawing area.
2) Repulsive Forces:
   - Nodes exert repulsive forces on each other to prevent overlap.
   - Force between two nodes i and j is given by Coulomb's Law: $F_{ij} = \frac{k^2}{d_{ij}}$, where k is a constant and d_{ij} is the distance between nodes.
3) Attractive Forces:
   - Connected nodes exert attractive forces to bring them closer.
   - Force between connected nodes i and j is given by: $F_{ij} = \frac{d_{ij}}{k}$, where k is a constant.
4) Updating Positions:
   - Nodes move based on the net forces acting on them.
   - The displacement of each node is calculated using Newton's second law: $m \cdot a = F$, where m is mass (assumed to be 1), a is acceleration, and F is the net force.
5) Temperature Control:
   - The temperature parameter is gradually decreased over iterations to control the movement of nodes.
   - As the temperature decreases, nodes move less, allowing the system to converge to a stable layout.

Output of algorithm:

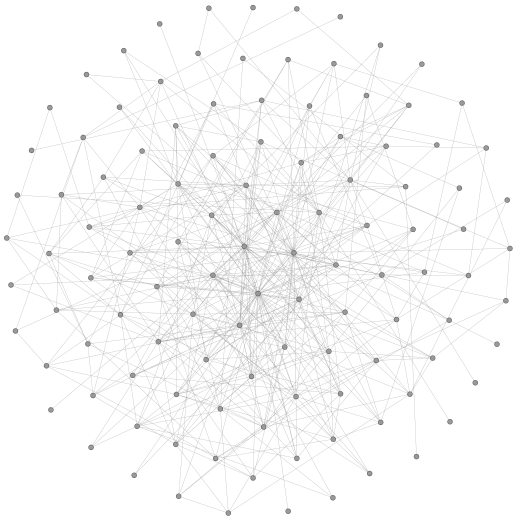

Fig. 54. Fruchterman Reingold algorithm applied to the given network

We observe the graph (Fig 54) taking a circular outline, with nodes having higher degrees occupying the center and nodes with lower degrees occupying the periphery. The nodes are evenly spaced and look neat. However, the edges seem quite cluttered, and connections between nodes are not obvious, especially at the center.

Mapping Degree with Color:

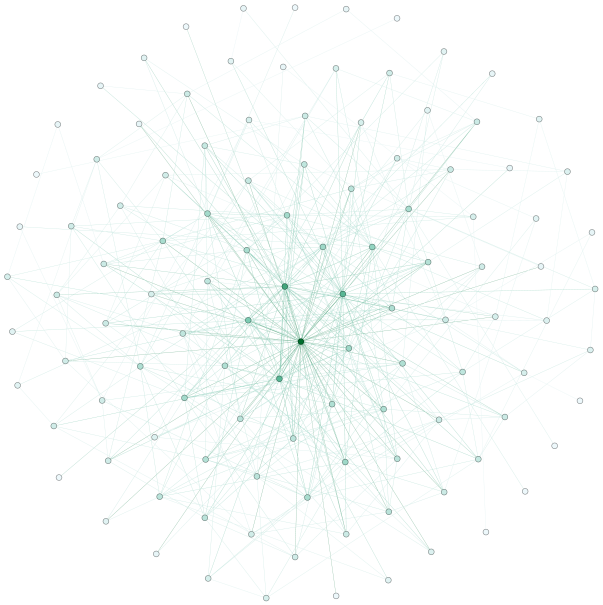

Fig. 55. The degree of each node being mapped to a monochromatic colormap

We can affirm our belief that nodes with a higher degree occupy the center (Fig 55), as the nodes in the center are apparent from the darker green color.

Looking at Nouns and Adjectives:

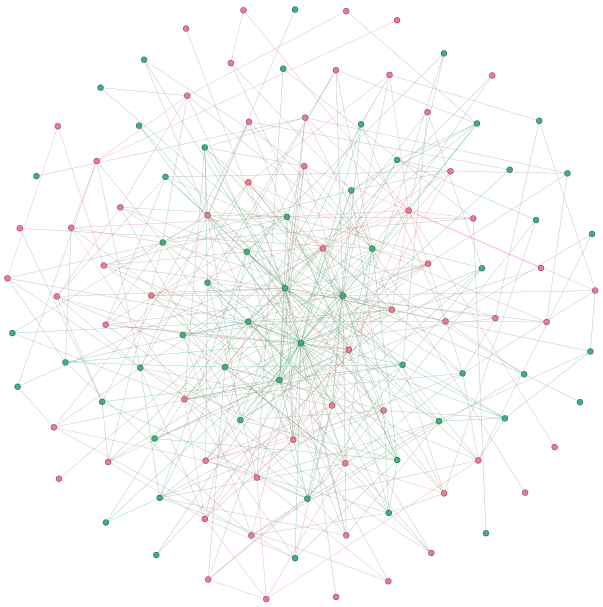

Fig. 56. Nodes representing nouns are colored pink, and nodes representing adjectives are colored green.

We observe (Fig 56 that although both nouns and adjectives seem to be at the center, nouns outnumber adjectives in the center, and vice versa in the periphery. We also observe that the graph is not bipartite; that is, nouns seem to occur with nouns, and adjectives occur with adjectives.

Visualizing with Node Labels

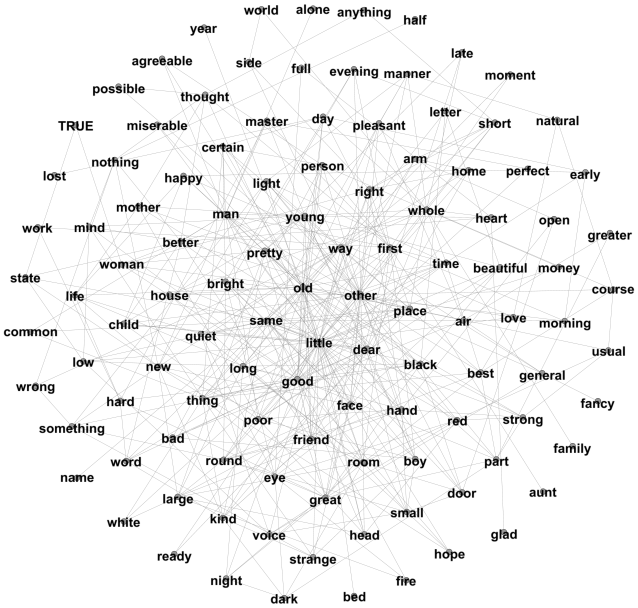

Fig. 57. Network in Fruchterman Reingold layout along with node labels.

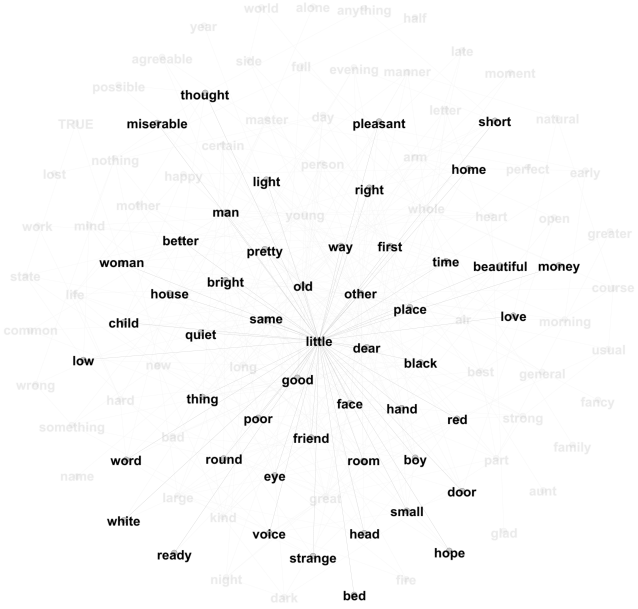

Fig. 58. The neighbors of the word 'little' highlighted

We see common adjectives and nouns at the center (Fig 57). The word 'little' seems to co-occur the most, having a degree of 49 (Fig 58).

4) The Force Atlas Algorithm:

Theory

1) Initialization:
   - Nodes are randomly placed in the drawing area.
2) Forces:
   - Attractive forces are applied between connected nodes, pulling them closer.
   - Repulsive forces act between all nodes, preventing overlap and distributing nodes evenly.
3) Scaling:
   - Forces are scaled based on the weights of the edges, giving more influence to stronger connections.
4) Barnes-Hut Approximation:
   - The algorithm often uses a Barnes-Hut approximation to efficiently compute long-range forces, reducing the computational complexity.
5) Temperature Control:
   - The system includes a temperature parameter that decreases over iterations, controlling the movement of nodes and allowing the layout to converge.

Output of the algorithm:

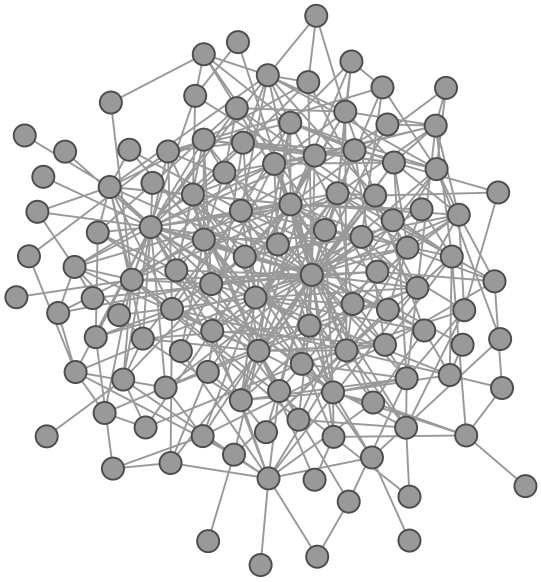

Fig. 59. Force Atlas algorithm applied to the given network

We observe (Fig 59) that nodes connected more are in the center of the graph, while those connected less are towards the periphery. The edges are very cluttered, and exact connections are not decipherable. In comparison to Fructherman Reingold, the nodes aren't as evenly spread out. The nodes are more densely located at the center than in the periphery.

Mapping Degree with Color:

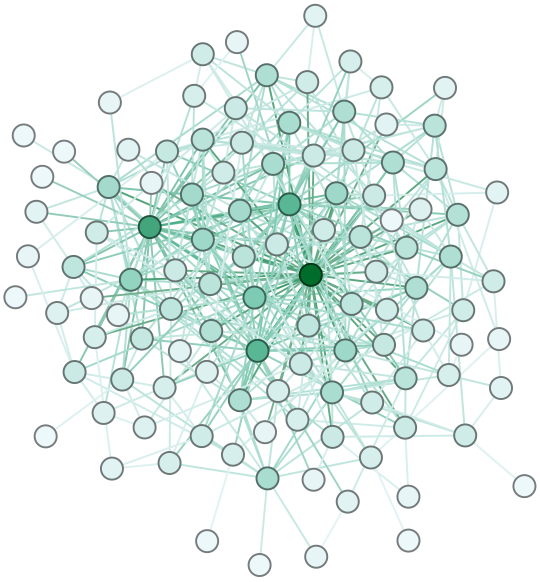

Fig. 60. The degree of each node being mapped to a monochromatic colormap

We can confirm our belief that nodes with a higher degree occupy the center (similar to Fruchterman Reingold), as the nodes in the center are apparent from the darker green color.

Looking at Nouns and Adjectives:

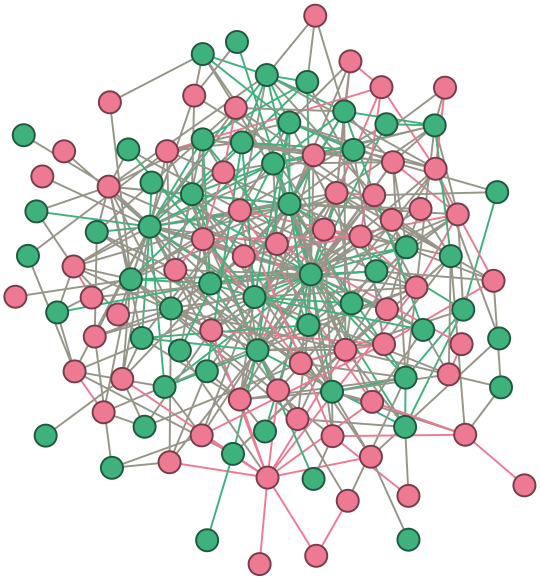

Fig. 61. Nodes representing nouns are colored pink, and nodes representing adjectives are colored green.

The observations are very similar to Fructherman Reingold algorithm. We observe (Fig 61) that although both nouns and

adjectives seem to be at the center, nouns outnumber adjectives in the center, and vice versa in the periphery. We also observe that the graph is not bipartite; that is, nouns seem to occur with nouns, and adjectives occur with adjectives.

Visualizing with Node Labels

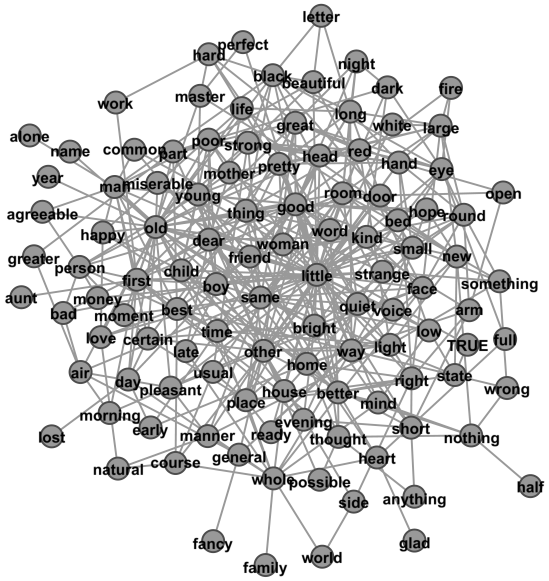

Fig. 62. Network in Force Atlas layout along with node labels.

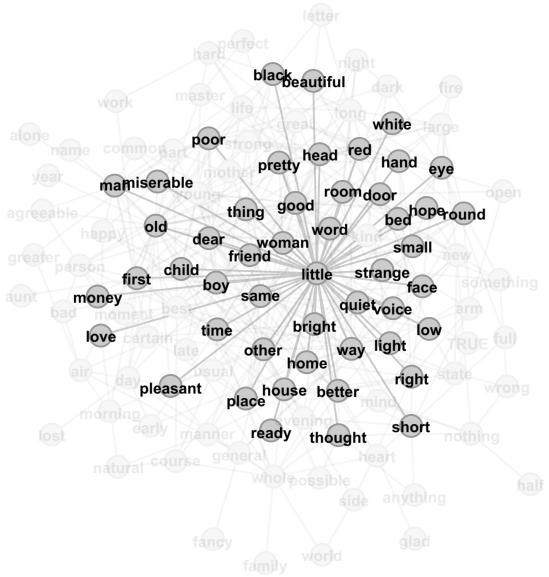

Fig. 63. The neighbors of the word 'little' highlighted

As in Fruchterman Reingold algorithm, we see common adjectives and nouns at the center (Fig 62). The word 'little' seems to co-occur the most, having a degree of 49 (Fig 63).

5) The Radial Axis Layout Algorithm:

Theory [11]

1) Node Placement:
   - Nodes are placed along radial axes emanating from a central point.
2) Hierarchy Representation:
   - The layout emphasizes hierarchical relationships, with parent nodes typically positioned closer to the center and child nodes arranged along the radial lines.
3) Angular Separation:
   - Nodes are evenly distributed along each radial line to provide clear separation and prevent overlap.
4) Connection Lines:
   - Edges between connected nodes are often represented as straight lines connecting nodes along the radial axes.
5) Angular Resolution:
   - The layout aims to maximize the angular resolution between nodes, making it easier to discern relationships in the hierarchical structure.

Output of the algorithm:

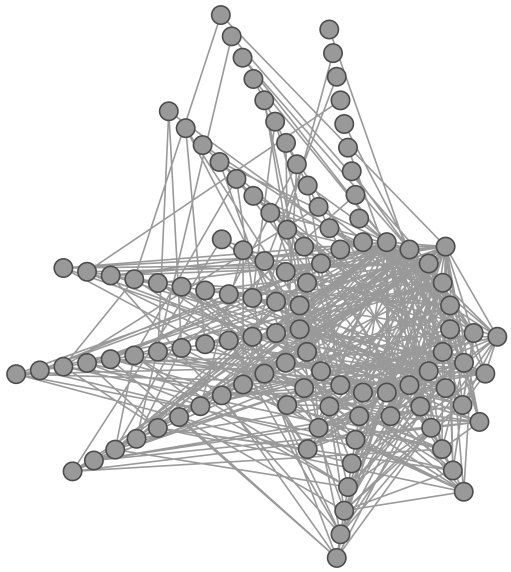

Fig. 64. Radial Axis Layout algorithm applied to the given network

The nodes are placed in a circular fashion, with some nodes stacked on others and thereby radially emanating outwards (Fig 64). The nodes are placed in the order of their degree, increasing in the counterclockwise direction, and nodes with the same degree are stacked on top of each other. The edges are very cluttered, implying no inherent hierarchy is present in the network in its present formation.

Mapping Degree with Color:

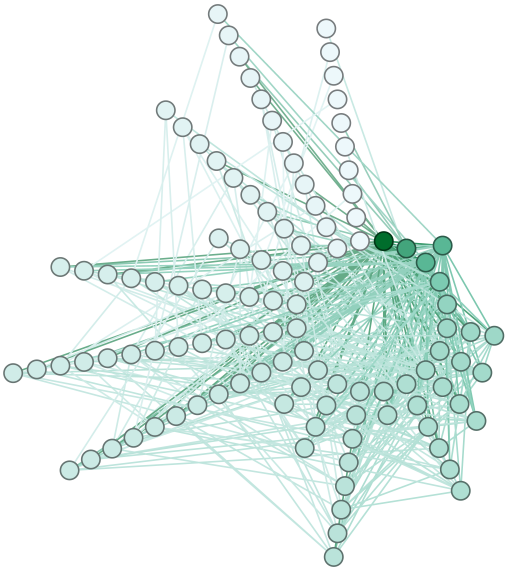

Fig. 65. The degree of each node being mapped to a monochromatic colormap

The degree increasing in the counterclockwise direction is more apparent (Fig 65).

Looking at Nouns and Adjectives:

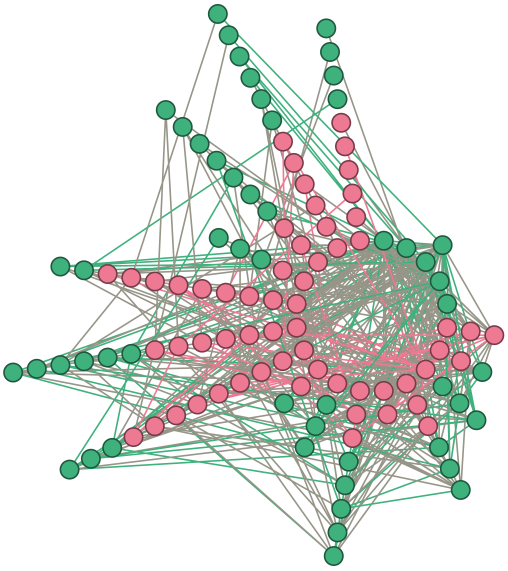

Fig. 66. Nodes representing nouns are colored pink, and nodes representing adjectives are colored green.

We can see that the number of adjectives with a higher degree seems to be more than nouns (Fig 66).

Visualizing with Node Labels

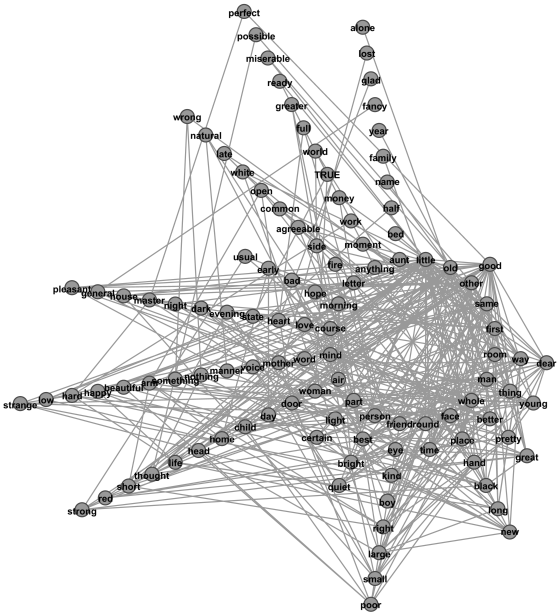

Fig. 67. Network in Force Atlas layout along with node labels. The graph has been slightly expanded for better readability of labels.

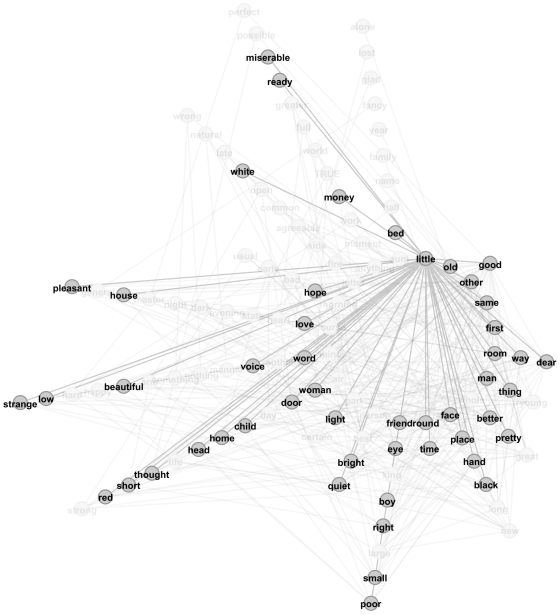

Fig. 68. The neighbors of the word 'little' highlighted

We see common adjectives and nouns at the top right (Fig 67). The word 'little' seems to co-occur the most, having a degree of 49 (Fig 68).

B. Parallel Coordinates Plot

1) Dataset: The dataset used is "Attacks on Health Care in Countries in Conflict (SHCC) Data" [12]. It provides various statistics pertaining to attacks on healthcare-related infrastructure across the world.

2) Data Processing: The data was first cleaned and processed using Pandas [13] by Sai Madhavan G for *CS732 - A1*. The resultant file had aggregated values for the number of incidents and the number of heath workers killed, kidnapped, injured, and arrested. Each data row in this file can be uniquely identified using (year, country) as the primary key. In other words, the data is aggregated in a country-wise manner, for all years (2017 - 2022).
When sparse data is further excluded from the chosen subset of data using Tableau Prep [14], we are left with data for the years {2017, 2019, 2020, 2021, 2022}. Finally, we group the data using two strategies:

- Country-wise grouping with aggregation across years, see Fig 69.
- Year-wise grouping with aggregation across countries, see Fig 70.

These two sets of data are used to visualize interactive Parallel Coordinates Plots which support user interactions such as brushing (filtering) and axes-reordering.

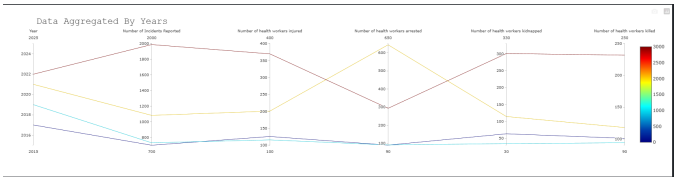

Fig. 69. PCP for Country-wise grouping with aggregation across years.

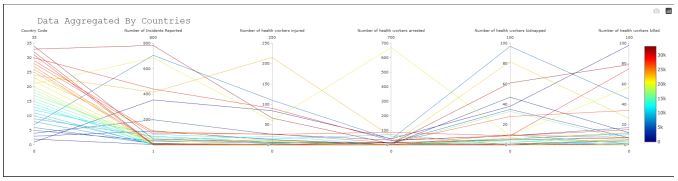

Fig. 70. PCP for Year-wise grouping with aggregation across countries.

3) Implementation: The codebase is primarily written in JavaScript, with parts of data processing done using Python. The final results are displayed on an HTML page, which can be viewed on a vsCode live server.
D3.js [24] is used to read the CSV data source, after which we make use of the Plotly.js [25] *Parcoords* [23] API to render an instance of the Parallel Coordinates Plot (PCP) visualization.
This process is done twice, once for year-wise grouping with aggregation across countries, and once for country-wise grouping with aggregation across years.

The API allows us to customize the color scale in order to allow different line traces to have different colors. For our purpose, we make use of the *Jet* colormap, and map each data index (which is either a year, or a country) to linearly spaced out colors. To do so, we introduced a new column *colorVal* during preprocessing. We chose a continuous colormap as it would make it easier to accommodate more data indices, if the data should grow in the future. It is important to note that the colors here are not indicative of any quantitative value, and they simply help differentiate between the data indices.
To represent categorical data indices, such as countries, we assign a numerical country code (a simple incremental id) to each country. Finally, the API also provides functionality to add annotations and labels on the final plot using the *layout* parameter.

4) User Interactions: The user can interact with the PCP visualizations primarily in one of two ways:

- Brushing (Filtering) is indicated by pink panes on the axes, see Fig 71, Fig 72, and Fig 73.

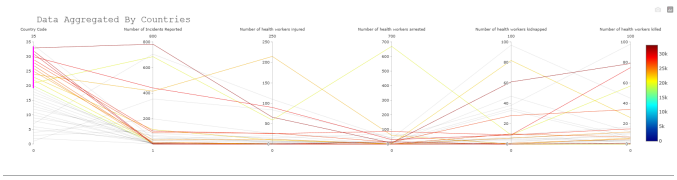

Fig. 71. Example of brushing over one axis.

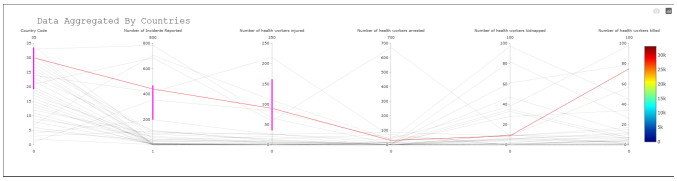

Fig. 72. First example of brushing over multiple axes.

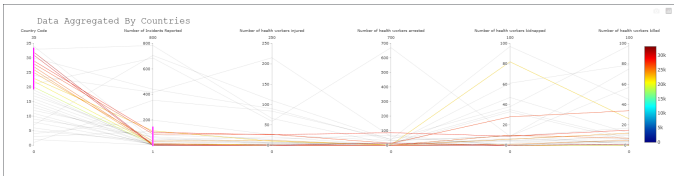

Fig. 73. Second example of brushing over multiple axes.

- Axes-Reordering, see Fig 74, and Fig 75, where the axis, to which a filter has been applied, has been reordered.

Brushing: The user can left-click and drag on one of the axes, selecting a subset of data points to filter and highlight them. If this action is repeated on multiple axes, what remains highlighted is an intersection of traces, which satisfy all constraints imposed by the user.
Axes-Reordering: The user can simply left-click on the name of an axis, and then drag it leftwards or rightwards to

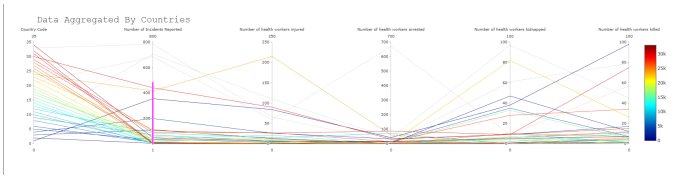

Fig. 74. Example of axis reordering - 1.

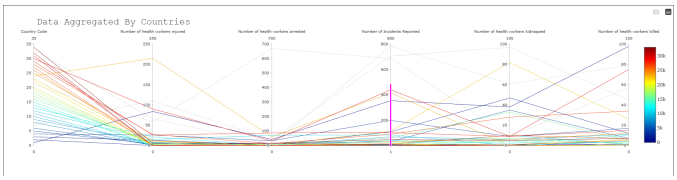

Fig. 75. Example of axis reordering - 2.

change its position. The axes are swapped appropriately to allow for a smooth transition.

We recognise that through the JavaScript browser API, it is possible to incorporate more complex user interactions. We hope to pursue this as future work.

5) Experiments: Several experiments were performed while using the Plotly.js *Parcoords* API. After going over different visualizations, which utilised various types of colormaps (sequential, diverging, continuous, discrete) and also incorporated annotations over the visualization to provide clarity, we came to the following conclusions:

- Annotations unnecessarily lead to a ”busy” visualization, often making it difficult to gain insights into the data.
- To accommodate new data being added in the future, a continuous color map is used. A discrete map would have limited the number of parallel traces in our visualization by the number of discrete volor bands defined.
- In an attempt to come up with a better implementation of PCP visualizations, our survey led to a GitHub Gist [16]. The incomplete, yet worthwhile effort produced the results seen in Fig 76, Fig 77, Fig 78. This implementation also allows for axes-reordering and brushing.

6) Observations and Inferences:

1) Based on the PCP for Country-wise grouping with aggregation across years, we can see that the majority of incidents are reported only by a handful of countries, namely Ukraine, Myanmar, DRC, OPT, and Afghanistan. This can be attributed to these regions either participating in wars or being subject to poor economic conditions and oppressive regimes.
2) **Dominican Republic of Congo** (DRC) reports one of the highest number of incidents, most of which are incidents of kidnappings. This can be inferred from Fig 79. These can be attributed to the region’s poor economic condition, which leads to kidnapping for ransom.
3) **Ukraine** also reports one of the highest number of

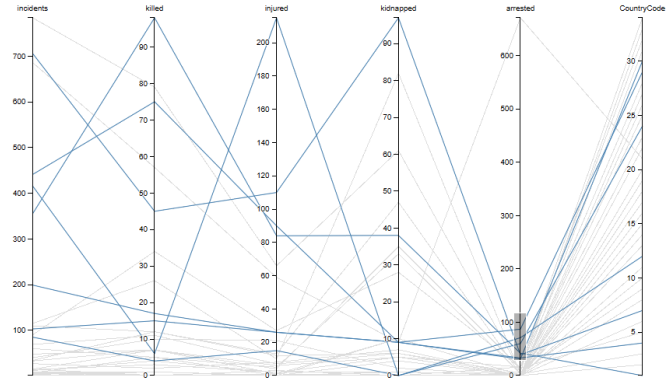

Fig. 76. Experimenting with PCP implementation - 1.

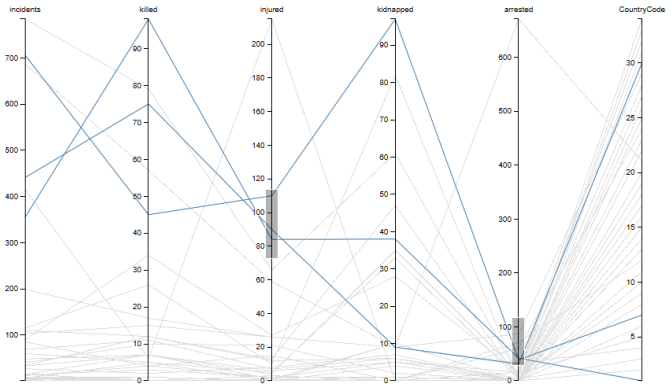

Fig. 77. Experimenting with PCP implementation - 2.

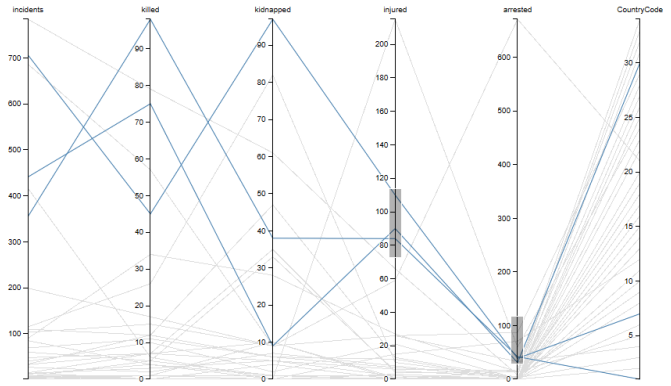

Fig. 78. Experimenting with PCP implementation - 3.

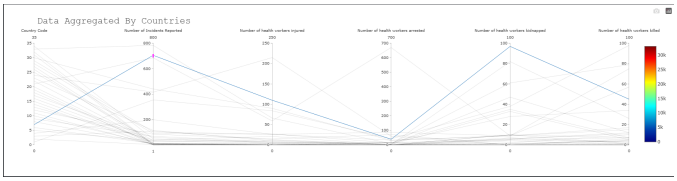

Fig. 79. Filtering to Show incident types in DRC.

incidents, which can be attributed to the country's recent war with Russia. In accordance to this fact, most of the incidents correspond to health workers being killed. This can be inferred from Fig 80.

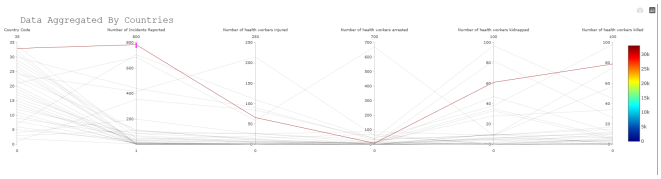

Fig. 80. Filtering to Show incident types in Ukraine.

C. Treemap Plot

1) Datase : The dataset used is "Attacks on Health Care in Countries in Conflict (SHCC) Data" [12]. It provides various statistics pertaining to attacks on healthcare-related infrastructure across the world.

2) Data Preprocessing : The data was cleaned and pre-processed. Initially, the incidents data for the year 2022 was available with individual dates and country information, which was then aggregated. Additionally, for a specific plot, a subset(columns) of the original 2022 incidents data was extracted, and subsequent plots were generated based on this subset.

3) Implementation: The treemap implementation is carried out in HTML using the AnyChart [21] and FusionChart [22] libraries. Initially, the data is extracted from a CSV file, and the user is prompted to input column values for treemap creation. The extracted data is then transformed into a hierarchical JSON format to visually represent the data hierarchy. Due to the limited inherent hierarchy in the data, only a few plots with hierarchical data were generated. Both AnyChart and FusionChart libraries were employed to account for missing features in each other. Additionally, various algorithms were tested using the FusionChart [22] library.

4) User Interaction : The plot offers users two additional interaction options, in addition to the provided HTML options:

- Users can select a specific block within the treemap to explore the inner hierarchy of the data, provided the data is hierarchical. Fig 81 shows the Countries and the number of incidents and Fig 82 After interacting (clicking) with the 'Ukraine' block
- If the FusionChart [22] library is employed, users can move the marker along the color scale to identify which blocks of the treemap fall within the color range defined by the markers (Fig 83).

5) Experiments : There were different experiments we tried:

- **Treemap Algorithms** There were different algorithms, viz. Squarified, Slice-and-dice (Horizontal, Vertical and Alternate) we tried; on both heirarchical and non-heirarchical data (Fig 84 to Fig 91). All of the mentioned

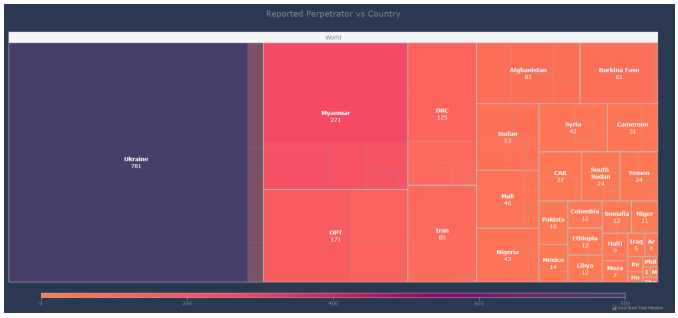

Fig. 81. Treemap for Heirarchical data

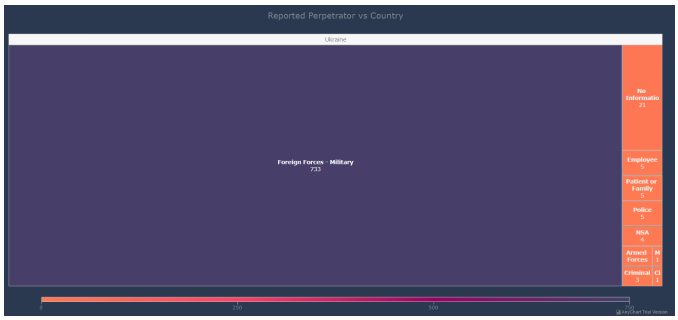

Fig. 82. User Interaction: after clicking on 'Ukraine' block

are shown in the figures below. Out of all of the Algorithms, the best one we found was the Squarified, which had the most readability.

- **Plot Settings:** For AnyChart, the colour of the blocks were picked automatically, provided that we had given initial gradient colours. For FusionChart, we had to manually assign a number to the blocks, which then picked from the gradient scale. We employed simple calcualtive colour formula, that assigns a number between the gradient scale proportional to how large the value of the block is. The best colours were found considering many factors viz. readability, etc.

6) Observation and Inferences : The treemap visualization presented below offers a comprehensive overview of the distribution of SHCC (attack on healthcare incidents) across various countries in 2022. This interactive treemap allows users to explore the hierarchical structure of incidents,

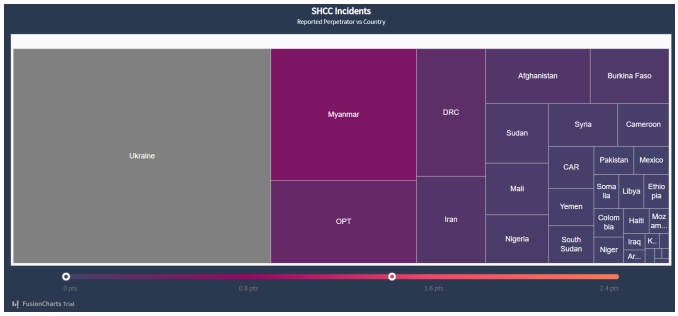

Fig. 83. User Interaction: Gradient Scale

Fig. 84. Slice and Dice Alternate Method for Treemap for Heirarchical Data

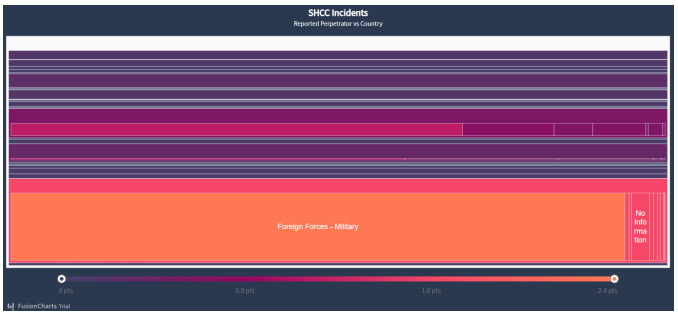

Fig. 85. Slice and Dice Horizontal Method for Treemap for Heirarchical Data

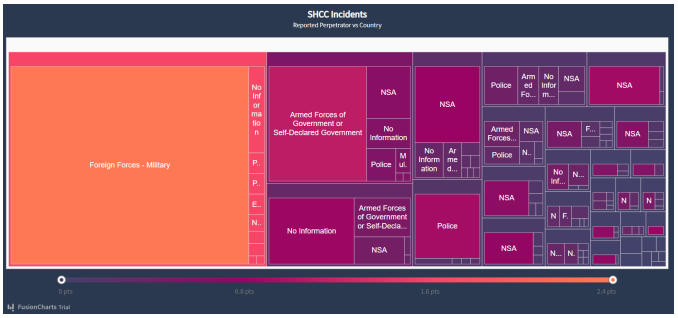

Fig. 86. Slice and Dice Squarified Method for Treemap for Heirarchical Data

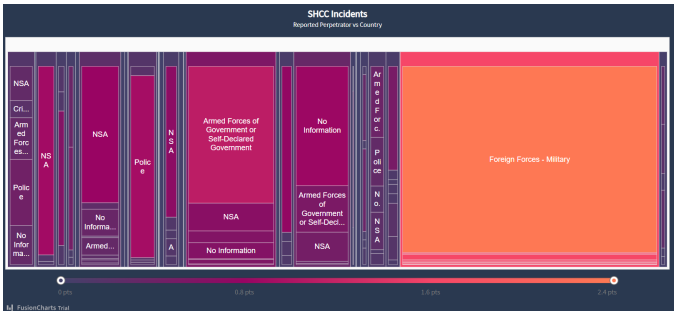

Fig. 87. Slice and Dice Vertical Method for Treemap for Heirarchical Data

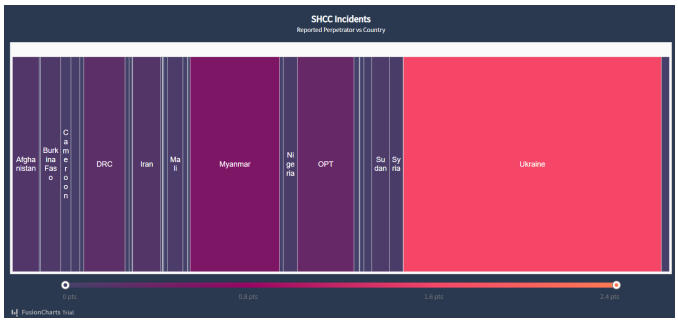

Fig. 88. Slice and Dice Vertical Method for Treemap for Non Heirarchical Data

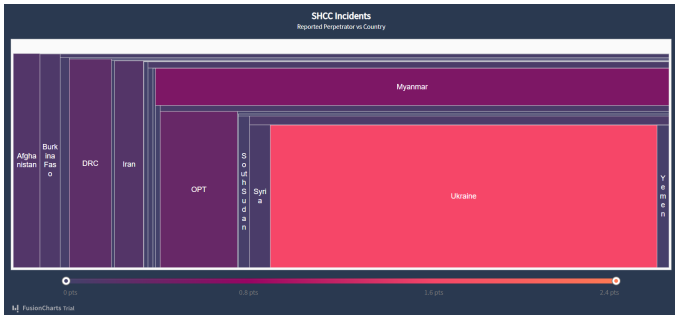

Fig. 89. Slice and Dice Alternate Method for Treemap for Non Heirarchical Data

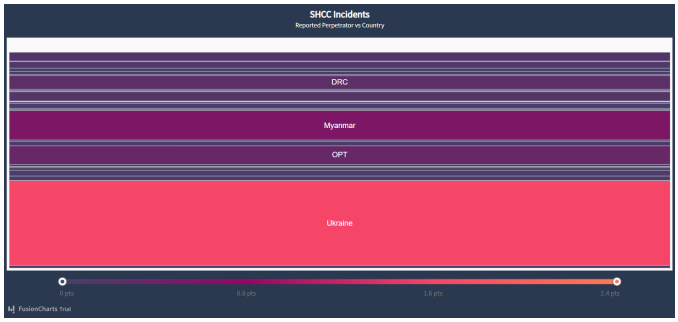

Fig. 90. Slice and Dice Horizontal Method for Treemap for Non Heirarchical Data

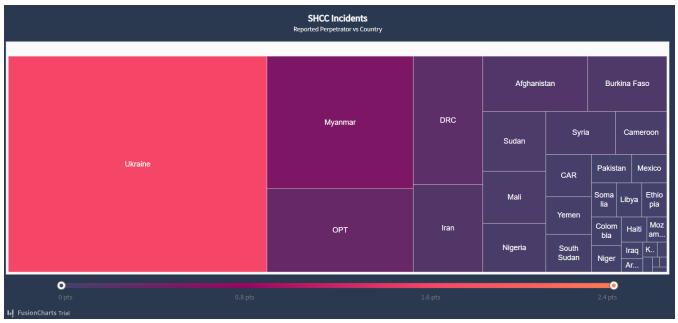

Fig. 91. Slice and Dice Squarified Method for Treemap for Non Heirarchical Data

providing detailed insights into the specific actors involved in each region.

The treemap efficiently conveys the scale and proportions of SHCC incidents in different countries. Each block in the treemap represents a country, with the size of the block proportional to the total number of reported incidents. Users can interactively navigate through the treemap to delve into specific countries and understand the nuances of the incidents within them.

- **Visualization 1- No. of Incidents and Reported Perpetrators in Countries:** The treemap visualization presented below offers a comprehensive overview of the distribution of Significant Hybrid Conflict and Crime (SHCC) incidents across various countries in 2022. This interactive treemap allows users to explore the hierarchical structure of incidents, providing detailed insights into the specific actors involved in each region.

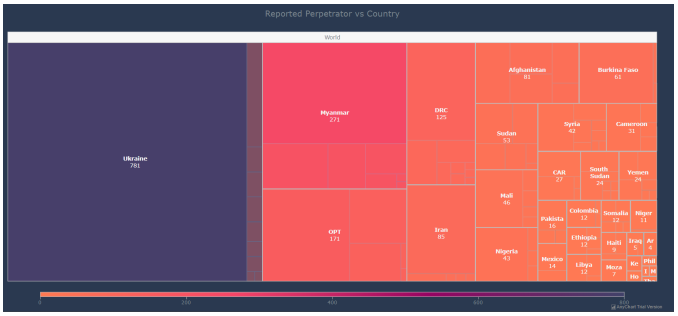

Fig. 92. Visualization 1- No. of Incidents and Reported Perpetrators in Countries:

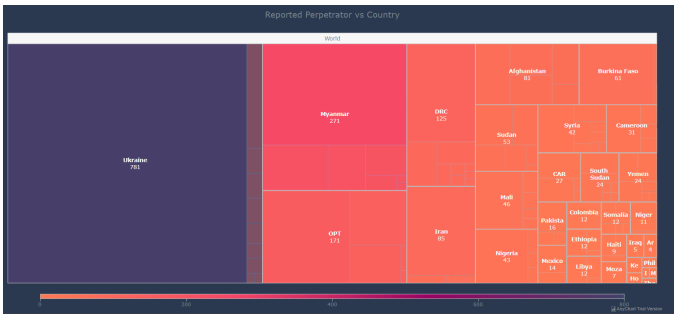

Fig. 93. Visualization 1- No. of Incidents and Reported Perpetrators in Ukraine:

Analysis: The treemap efficiently conveys the scale and proportions of SHCC incidents in different countries. Each block in the treemap represents a country, with the size of the block proportional to the total number of reported incidents. Users can interactively navigate through the treemap to delve into specific countries and understand the nuances of the incidents within them.

For instance, in Ukraine, a total of 781 SHCC incidents were reported in 2022. Upon entering the Ukrainian treemap node, it becomes apparent that the majority of these incidents, precisely 733, were caused by Foreign Military Forces. This detailed breakdown provides valuable context, allowing for a nuanced understanding of the dynamics contributing to the overall incident count.

Similarly, Afghanistan recorded a total of 81 reported incidents in 2022. By exploring the treemap node for Afghanistan, users can uncover that the Afghanistani Police were the primary contributors, accounting for 27 incidents. This insight into the specific actors involved adds depth to the analysis, facilitating a more targeted approach to addressing security concerns.

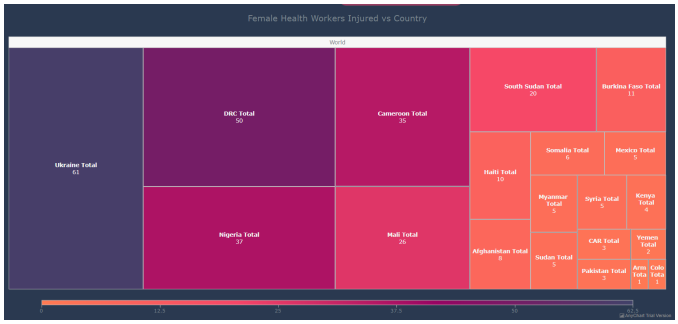

Fig. 94. No. Female Health Workers Injured in Countries

- **Visualization 2- No. of Incidents of Female Health Workers Injured in Countries:** The treemap visualization in Fig. 94 delves into the distribution of incidents affecting female health workers in 2022 across different countries. The treemap efficiently conveys the varying levels of incidents impacting female health workers across different countries. Each block in the treemap represents a country, and its size corresponds to the total number of reported incidents. For instance, while the proportion of overall incidents in (Vis 1) may be lower in African nations, a closer examination within the treemap nodes may unveil that the impact on female health workers is disproportionately significant. This nuanced understanding is crucial for identifying regions where female health workers face elevated risks and where targeted interventions may be necessary.

Fig. 95. No. of Incidents of Obstruction to healthcare in Countries

- **Visaulization 3- No. of Incidents of Obstruction to healthcare in Countries** The treemap visualization in Fig. 95 explores the obstruction of healthcare services

globally, emphasizing the accessibility of health facilities in different countries. Contrary to previous visualizations where Ukraine featured prominently, the absence of reported incidents in this healthcare obstruction treemap suggests that health facilities were generally accessible to the public. Instead, incidents are concentrated in countries facing political or financial challenges. This aligns with findings from earlier visualizations, where Ukraine incidents were linked to Foreign Armed Forces, validating the inference that external factors often contribute to healthcare obstruction.

In summary, the treemap analysis of healthcare obstruction reaffirms the correlation between incidents and regions experiencing political or financial instability. The absence of reported incidents in Ukraine supports the inference that external forces, such as Foreign Armed Forces, often play a role in obstructing healthcare services in politically unstable regions. This succinct visualization sheds light on the global landscape of healthcare accessibility and the impact of external factors on healthcare obstruction.

III. Authors' Contributions

Sai Madhavan G: (Quiver Plot and Node-Link Diagrams)
Vidhish Trivedi: (Color Map and Parallel Coordinates Plot)
Krutik Patel: (Contour Map and Tree Map)

Each team member worked independently on their assigned task, including all aspects of data analysis, visualization creation, interpretation, and description, unless specified otherwise. The collaborative effort of all team members contributed to the comprehensive analysis presented in this report.

References

[1] https://las.incois.gov.in/las/UI.html
[2] https://unidata.github.io/netcdf4-python/
[3] https://numpy.org/
[4] https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html
[5] https://matplotlib.org/basemap/
[6] http://konect.cc/networks/adjnoun_adjacency/
[7] https://en.wikipedia.org/wiki/David_Copperfield
[8] https://gephi.org/
[9] https://www.nltk.org/api/nltk.tag.perceptron.html#nltk.tag.perceptron.AveragedPerceptron
[10] Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. Software: Practice and Experience, 21(11), 1129-1164. [DOI: 10.1002/spe.4380211102](https://doi.org/10.1002/spe.4380211102)
[11] Jacomy, M., Venturini, T., Heymann, S., & Bastian, M. (2014). ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. PloS one, 9(6), e98679. [DOI: 10.1371/journal.pone.0098679](https://doi.org/10.1371/journal.pone.0098679)
[12] https://data.humdata.org/dataset/shcchealthcare-dataset ; Note that this is the same dataset as the one used for CS732 - Assignment 1.
[13] https://pandas.pydata.org/
[14] https://www.tableau.com/products/prep
[15] https://www.britannica.com/science/Indian-monsoon
[16] https://gist.github.com/jasondavies/1341281
[17] https://docs.xarray.dev/en/stable/
[18] https://unidata.github.io/netcdf4-python/
[19] https://pypi.org/project/imageio/
[20] https://github.com/SciTools/cartopy
[21] https://www.anychart.com/
[22] https://www.fusioncharts.com
[23] plotly.com/javascript/parallel-coordinates-plot/
[24] https://d3js.org/
[25] https://plotly.com/