***SUMMER INTERNSHIP PROJECT REPORT ON***

# “DISASTER RELIEF AND RESCUE MANAGEMENT SYSTEM”



# Project In-Charge Project Mentor

|  |  |
| --- | --- |
| **MR. RAJEEV KUMAR JHA**  **SCIENTIST-E NATIONAL**    **INFORMATICS CENTRE**  **BIHAR STATE CENTRE,**  **PATNA**  **Submitted by** | **MR. RAKESH KUMAR**  **SCIENTIST-E NATIONAL**  **INFORMATICS CENTRE**  **BIHAR STATE CENTRE,**  **PATNA** |

ARYAN N K ARYA

B.TECH (CSE)

BIRLA INSTITUTE OF TECHNOLOGY, MESRA

APPLICATION REF. NO. – BIT/T&P/3653

DIGITAL GOVERNMENT RESEARCH CENTRE, PATNA

NATIONAL INFORMATICS CENTRE, PATNA, BIHAR- 800015



**DIGITAL GOVERNMENT RESEARCH CENTRE, PATNA**

**NATIONAL INFORMATICS CENTRE, PATNA, BIHAR- 800015**

# CERTIFICATE

This is to certify that ARYAN N K ARYA of BTECH, CSE has presented a project titled "Bihar ColdWave Dashboard and Disaster Incident Dashboard" under the Digital India Summer Internship program during the period 7th May 2025 to 7th July 2025 at the Digital Government Research Centre, Patna.

## Project In-charge Project Mentor

Rajiv Kumar Jha Rakesh Kumar

Scientist E Scientist E

National Informatics Centre National Informatics Centre

Bihar State Centre Bihar State Centre

Patna Patna

## DECLARATION

I hereby declare that the project report entitled "Bihar ColdWave Dashboard" and “Bihar Disaster Incident Dashboard” is my original work. This written submission represents my ideas in my own words and wherever others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or falsified any idea, data, or fact in my submission.

## ACKNOWLEDGEMENT

I would like to express my deep and sincere gratitude and regards to the people who have rendered their support, supervision, guidance and sincere help during

the course of this project work. I would like to express my profound gratitude to

Mr. Rajeev Kumar Jha, HOD, DGRC, Patna, NIC for giving me the opportunity to work as a Summer Intern at Digital Government Research Centre, Patna. I would like to express my special thanks to my mentor Mr.Rakesh Kumar,

Director (IT) and Ms. Nagma Parween, Scientific Tech. Asst. -B, for their time and efforts, they provided throughout the internship. Their useful advice and suggestions were really helpful for me during the project’s completion. In this aspect, I am extremely grateful to them.

I would like to thank all the employees and workers at Digital Government Research Centre, Patna for cooperating with me and giving me a positive work environment.

I deeply acknowledge the love, cooperation and the moral support extended by my family members and friends during this period.

# ABSTRACT

These web-based data visualization tools are specifically designed to enhance the capabilities of the Emergency Operations Center within the Government of Bihar. They provide a centralized, interactive platform for monitoring and analyzing various types of natural and human-induced incidents across the state. Given Bihar's vulnerability to diverse emergencies—ranging from severe cold waves to a spectrum of other critical incidents—these dashboards serve to deliver real-time and historical data to government officials, emergency responders, and district administrators. Leveraging Python with the Streamlit framework for the user interface, Pandas for data manipulation, and Plotly for rich, interactive visualizations, they enable data-driven decision-making, optimize resource allocation, and strengthen overall emergency response and planning. By consolidating disparate incident and environmental data into intuitive visual formats, these systems significantly improve the efficiency, transparency, and accountability of disaster and incident management operations, serving as powerful tools for situational awareness and strategic intervention across all districts and blocks of Bihar.

## TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Organization Profile

1.2 The Digital India Initiative

CHAPTER 2: PROJECT BACKGROUND

2.1 Problem Description

2.2 About the Project

CHAPTER 3: PROJECT OBJECTIVE

CHAPTER 4: TOOLS AND TECHNOLOGY

CHAPTER 5: SYSTEM ARCHITECTURE

5.1 Data Layer

5.2 Application Logic Layer

5.3 Presentation Layer

CHAPTER 6: PROJECT FEATURES AND WORKING

6.1 Main Dashboard Layout

6.2 Sidebar and Filtering

6.3 KPI Card Navigation and Display

6.4 Interactive Geo-Map

6.5 District-wise Data Visualization 6.6 Drill-Down and Trend Analysis

CHAPTER 7: ADVANTAGES

CHAPTER 8: LIMITATIONS

CHAPTER 9: CONCLUSION

CHAPTER 10: FUTURE SCOPE

CHAPTER 11: REFERENCES

CHAPTER 12: APPENDIX 12.1 Full Source Code **CHAPTER 1: INTRODUCTION**

## 1.1 Organization Profile

### 1.1.1 National Informatics Centre (NIC)

The National Informatics Centre is a part of the Indian Ministry of Electronics and Information Technology. It has its headquarters in New Delhi. It is the premier science & technology organization of the Government of India in Informatics Services and Information & Communication Technology (ICT) applications. It enables the improvement of government services and also maintains transparency in these services. Almost all Indian-government websites are developed and managed by NIC. **1.1.2 Digital Government Research Centre (DGRC)**

National Informatics Center (NIC), the Ministry of Electronics and Information

Technology (MeitY), launched the "Digital Government Research Centre

(DGRC)" in Patna on 2nd March, 2017. The Digital Government Research Centre (DGRC) focuses on developing tools and technology for digital government initiatives and also helps to introduce ICT solutions in improving service delivery to the common man across the nation.

## 1.2 The Digital India Initiative

The Digital India campaign was launched by the Government of India in 2015 to ensure that government services are made available to citizens electronically by improved online infrastructure and by increasing internet connectivity. The vision of the Digital India program is centered on three key areas - digital infrastructure as a utility to every citizen, governance & services on demand, and digital empowerment of citizens.

## CHAPTER 2: PROJECT BACKGROUND 2.1 Problem Description

Bihar, a state prone to various natural and human-induced calamities, faces significant challenges in comprehensive disaster and incident management. Beyond recurrent events like floods, the state regularly contends with severe cold waves, road accidents, lightning strikes, fires, and other critical incidents. Each of these events, whether seasonal or unpredictable, can lead to substantial human loss, displacement, damage to infrastructure, and severe socio-economic disruption.

Effective management of such diverse and large-scale emergencies necessitates access to timely, accurate, and consolidated information. Traditionally, data pertaining to incident impact—including affected populations, casualties, resource deployment, and damages—is often collected and compiled through disparate, manual, or localized reporting mechanisms. This decentralized approach frequently results in slow data aggregation, potential inaccuracies, and a fragmented view of the ground situation. State-level decision-makers and emergency responders find it challenging to obtain a clear, real-time, consolidated understanding of complex, multi-faceted incidents or widespread environmental hazards like cold waves. The absence of a centralized, interactive, and comprehensive data management platform directly impedes rapid assessment, efficient resource allocation, proactive response planning, and effective inter-departmental coordination, thereby impacting the overall efficacy of public safety initiatives.

## 2.2 About the Project

Addressing the critical challenges in comprehensive disaster and incident management, these projects introduce advanced, web-based dashboards designed for the Emergency Operations Center within the Government of Bihar. These platforms provide a unified solution for visualizing and analyzing data related to both cold waves and a diverse range of other incidents (such as road accidents, lightning, and fires) across the state.

Built with modern data science and web development tools, including Python, Streamlit, Pandas, and Plotly, the dashboards connect to a centralized SQL Server database. This robust backend enables them to fetch and display critical information in an intuitive, interactive, and real-time format. The primary aim of these tools is to empower the Emergency Operations Center, district administrations, and other government bodies with the data-driven insights essential for effective, timely, and proactive response to various emergencies. By consolidating disparate data, facilitating multi-level filtering, and providing comprehensive visualizations, these dashboards enhance situational awareness, optimize resource allocation, and strengthen the state's overall capacity for public safety and incident mitigation.

## CHAPTER 3: PROJECT OBJECTIVE

The primary objective of these projects is to design and develop interactive and user-friendly dashboards for comprehensive disaster and incident management across Bihar. The specific goals are:

* To establish a centralized and authoritative data source for information related to cold waves and various other incidents, integrating data from diverse reporting channels across districts and blocks.
* To provide real-time or near-real-time visualization of Key Performance Indicators (KPIs) relevant to both cold wave impacts (e.g., affected population, shelter usage, relief distribution) and general incident statistics (e.g., total incidents, deaths, injured counts).
* To develop interactive visualizations, including geographical representations and time-series charts, that visually represent the extent and trends of different disaster and incident types across the state and at granular district/block levels.
* To enable in-depth comparative analysis across districts and incident types to aid in prioritizing resource allocation, identifying high-risk areas, and informing strategic interventions.
* To build robust, scalable, and high-performance applications using Python and the Streamlit framework, ensuring ease of maintenance, future enhancements, and compatibility across various devices.
* To significantly improve the overall efficiency, transparency, and effectiveness of emergency response, public safety initiatives, and data-driven decision-making processes within the state's disaster management framework.

## CHAPTER 4: TOOLS AND TECHNOLOGY

The dashboard was developed using a suite of powerful, open-source Python libraries, each serving a specific purpose.

* **Python(3.12.6):** A versatile, high-level programming language chosen for its extensive data science ecosystem and readability. It serves as the backbone of the entire application.
* **Streamlit(1.45.1):** An open-source app framework used to build and deploy the web-based user interface for the dashboard. Its simplicity and fast development cycle make it ideal for creating data-centric applications. It handles the rendering of all UI components, including buttons, charts, and layout.
* **Pandas(2.2.3):** A fundamental library for data analysis and manipulation. It is used for fetching data from the database into a structured DataFrame, cleaning data types, handling missing values, and performing complex grouping and aggregation operations required for the visualizations.
* **Plotly(6.0.1):** A powerful graphing library used to create high-quality, interactive visualizations.Bihar ColdWave Dashboard uses Plotly for time series analysis and geographic distribution, while Bihar Incident Dashboard uses it for KPI gauges, trend analysis, and hierarchical data visualization.
* **Pyodbc(5.2.0):** An open-source Python module that provides a simple and efficient interface to connect to and query ODBC (Open Database Connectivity) databases. It acts as the bridge between the Python application and the Microsoft SQL Server database, allowing the script to execute SQL queries and retrieve flood data.

## CHAPTER 5: SYSTEM ARCHITECTURE

The application is designed with a three-tier architecture to separate concerns and improve maintainability.

## 5.1 Data Layer

* **Database:** Microsoft SQL Server acts as the central repository for all flood-related data. Data is assumed to be stored in tables like ColdWave Details and Hazard Report.
* **Connection:** The pyodbc library handles the connection to the SQL Server database. Connection details are securely stored in a Dashboard1.toml and Dashboard2.toml file for the ColdWave Dashboard and the Disaster Incident Dashboard respectively.
* **Data Retrieval:** The load\_data\_from\_db function executes a SQL query to fetch the raw data, which is then loaded into a Pandas DataFrame. Caching is used to prevent re-fetching data on every user interaction.

## 5.2 Application Logic Layer

This layer, written in Python, contains the core business logic.

* **Data Processing:** The Pandas library is used to process the raw data. This includes cleaning, filtering based on user input, and aggregating data for visualization.
* **State Management:** Streamlit's session state (st.session\_state) is used as a memory for the application, maintaining user selections across interactions.
* **Helper Functions:** Utility functions handle repetitive tasks like calculating KPI values, encoding images, and generating filter lists.

## 5.3 Presentation Layer

This is the user interface built with Streamlit components.

* **Framework:** Streamlit renders all visual components and handles the application's reactive nature.
* **Layout:** The UI is structured into a sidebar for controls and a main area for displaying information.
* **Components:** Widgets like st.selectbox and st.date\_input allow users to filter data. st.plotly\_chart displays interactive graphs. st.button is customized with CSS to act as clickable KPI cards.
* **Styling:** Custom CSS is injected to create a polished and professional look.

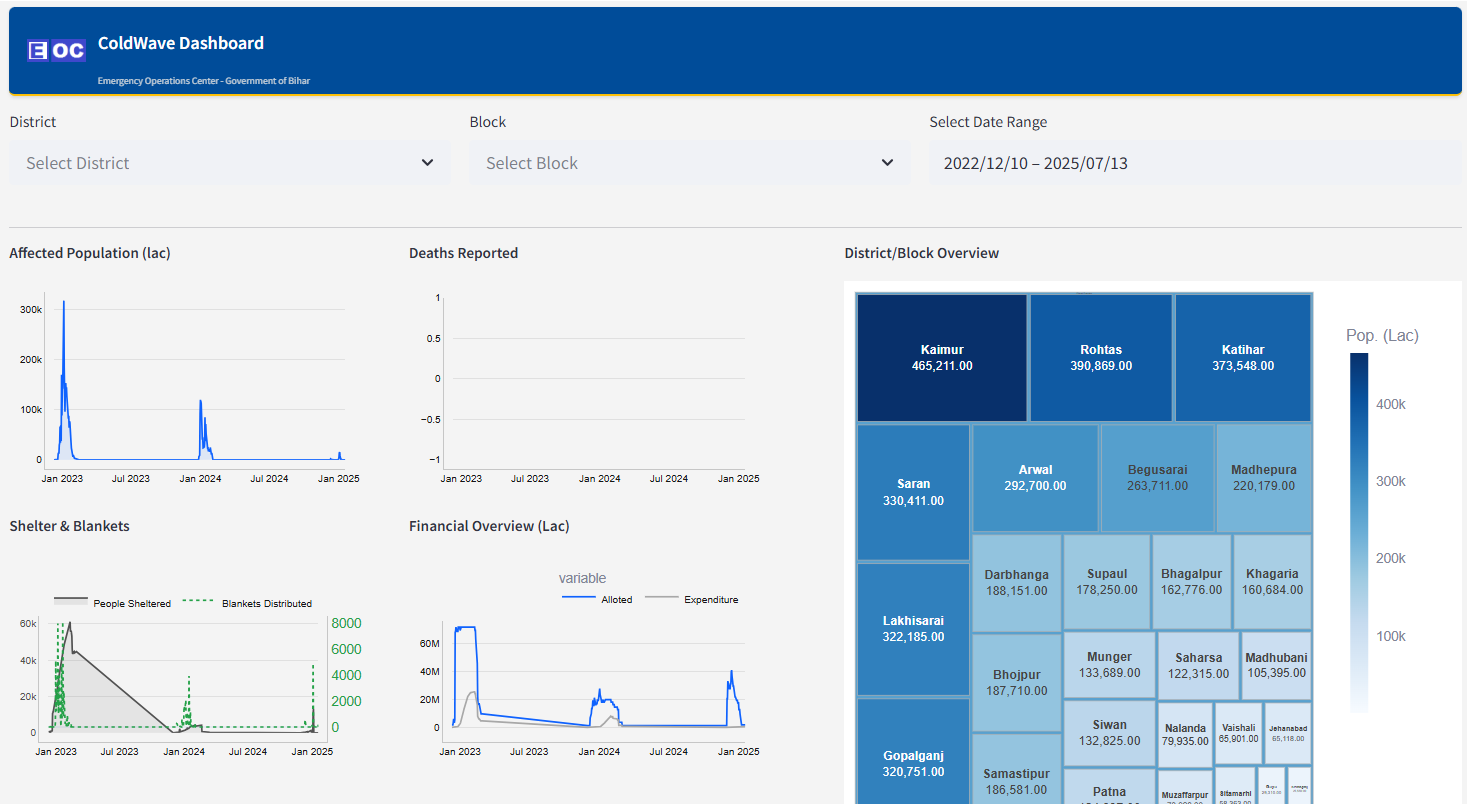
## CHAPTER 6: PROJECT FEATURES AND WORKING

The dashboards collectively offer a multi-faceted view of critical cold wave impacts and diverse incident scenarios across Bihar through their rich interactive components.

## 6.a Bihar ColdWave Dashboard

## 6.a.1 Main Dashboard Layout

The Cold Wave Dashboard offers a clear and intuitive layout for managing cold wave incidents in Bihar. It features a prominent interactive filter section at the top for selecting date ranges, districts, and blocks. Below the filters, the dashboard presents dynamic visualizations, including time-series line charts for metrics like Affected Population and relief efforts (People Sheltered, Blankets Distributed), financial overviewcharts, and a hierarchical tree map for district and block-wise impact analysis. This structure ensures quick assessment and data-driven decision-making for emergency response.



## 6.a.2 KPI Card Navigation and Display

Below the main filter section, a prominent row of cards is dedicated to displaying critical Key Performance Indicators (KPIs) for cold wave management. These cards provide immediate, aggregated numerical summaries of the situation.

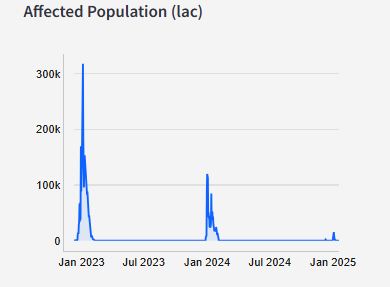
* **Comprehensive Metrics:** The KPIs showcased include the Affected Population (in lakhs), Human Death Count, number of Rain Baseras (night shelters) established, Allotted Amount and Expenditure Amount (in lakhs), Blanket Distribution, People in Rain Basera, Wood Burn (in kg), and the number of Bonfire Places.
* **Visual Presentation:** Each KPI is presented clearly within its dedicated card, offering a quick overview of the current state of cold wave impact and relief efforts.
* **Interactive Hover Effect:** When users hover over these KPI cards, a subtle transition effect occurs, enhancing the user experience and drawing attention to the specific metric without requiring a click action. These cards are designed for static display of key figures and are not interactive for filtering other visualizations.



## 6.a.3 Interactive Charts

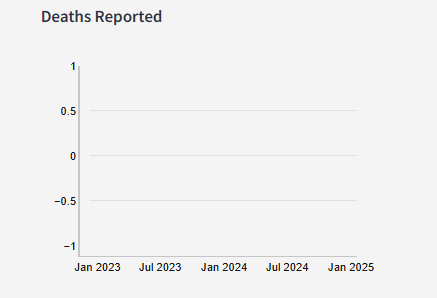
**1. Affected Population Trend Line Chart**

This line chart provides a clear historical view of the Affected Population due to Cold Wave in Bihar over time. It allows users to observe the fluctuating numbers of individuals impacted by cold weather conditions across different periods. The chart highlights trends, enabling quick identification of peak impact periods and the overall trajectory of cold wave effects on the population. Tooltip information on hover provides precise figures for specific dates.



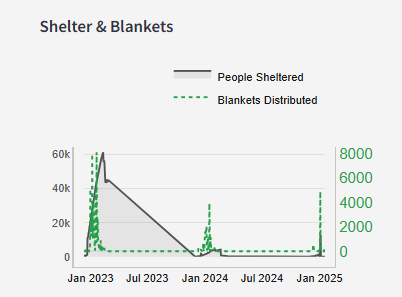
**2. Death Chart (No Deaths Recorded)**

This chart is specifically dedicated to visualizing the number of deaths over time attributed to cold wave incidents. As indicated, this chart is often empty, which signifies a successful outcome of no reported fatalities due to cold waves during the tracked period. While currently showing no data, its presence on the dashboard is crucial for quickly ascertaining the human cost and confirming the effectiveness of preventative and relief measures.



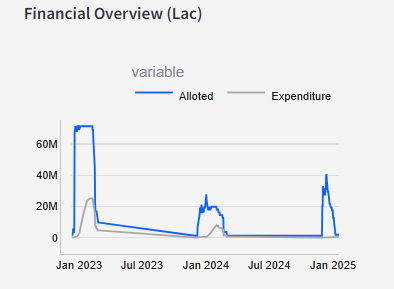
**3. People Sheltered and Blankets Distributed Over Time**

This dual-purpose chart tracks two critical relief metrics simultaneously: the number of People Sheltered in temporary accommodations (like Rain Baseras) and the quantity of Blankets Distributed to the affected population over time. This visualization allows for direct assessment of the outreach and immediate support provided to those in need, showing the progress of these vital relief efforts. Tooltips provide specific counts for both metrics on particular dates.



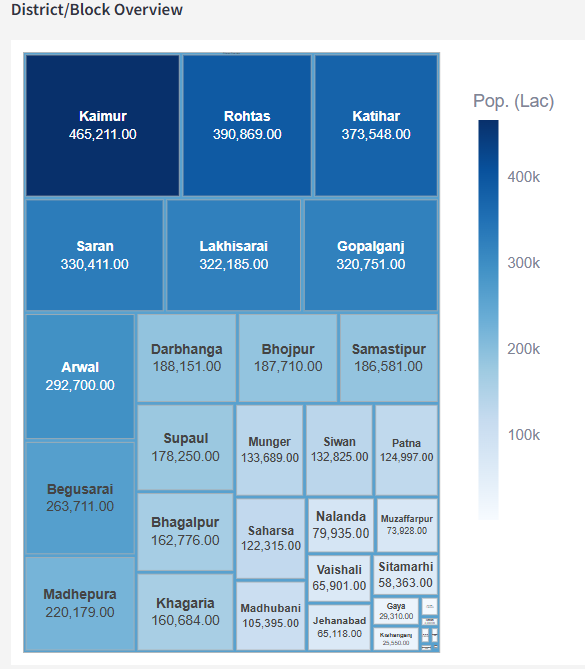
**4. Amount Allotted and Expended Over Time**

This chart provides a transparent financial overview by displaying the Amount Allotted for cold wave relief against the Amount Expended over time. It is crucial for financial oversight, allowing administrators to monitor budget utilization, track spending patterns, and ensure that funds are being effectively deployed for relief operations. Tooltip details provide exact figures for both allocated and expended amounts on specific dates.



**5. District-wise Affected Population Tree Chart**

This interactive Tree Chart offers a hierarchical visualization of the number of people affected due to Cold Wave, broken down District-wise. It provides an immediate sense of which districts are most impacted. A key feature of this chart is its dynamic response to filtering: when a specific District is selected in the filter, the chart transforms to show the population affected within that District's various Blocks. This drill-down capability is exemplified when, for instance, Patna district is selected, revealing the affected population distribution across its individual blocks. This allows for granular, location-specific planning and response.



**6.b Bihar Disaster Incident Dashboard**

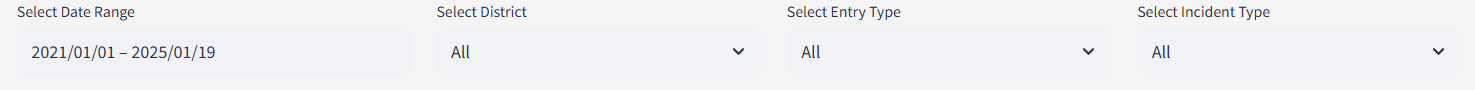
**6.b.1 Main Dashboard layout**

The Incident Dashboard is meticulously structured to provide a comprehensive and immediate overview of various incidents and associated human losses across Bihar. Upon loading, the layout prioritizes critical information, beginning with three prominent Key Performance Indicator (KPI) gauge charts at the top, visually representing Total Incidents, Total Deaths, and Total Injured, offering a quick snapshot of the state-wide situation. Immediately below these gauges, the dashboard features an interactive casualty section with colored cards that detail death counts by specific incident types, allowing for quick identification of the deadliest categories. The lower half of the dashboard is dedicated to time-series and geographical visualizations: an interactive daily deaths chart tracks fatalities for the current month, complemented by a monthly deaths chart providing yearly aggregations and seasonal patterns. Finally, a comprehensive district and block treemap spans the bottom, illustrating the geographical distribution of incidents and their types at a granular level. This intuitive and hierarchical layout, combined with a flexible filter section (not fully visible in this specific view but integral to the dashboard's functionality), ensures that users can efficiently navigate, analyze, and gain actionable insights into incident management.

**6.a.2 Interactive Filters Section**

While not a chart, the dashboard incorporates a robust filter section at the top, which drives the data displayed across all charts and components. This section includes:

* **Date Range**: Allows precise selection of start and end dates using a calendar picker.
* **District Filter**: A dropdown menu to select a specific district in Bihar, or view data for "All" districts.
* **Entry Type Filter**: Enables filtering by data verification status (e.g., All, Final, Verified, Unknown).
* **Incident Type Filter**: Allows users to narrow down data to specific incident categories (e.g., All, Drowning, Road Accident, Lightning, Fire, Heat Wave, Snakebite). These filters provide comprehensive control over the data view, ensuring highly customizable and relevant insights.



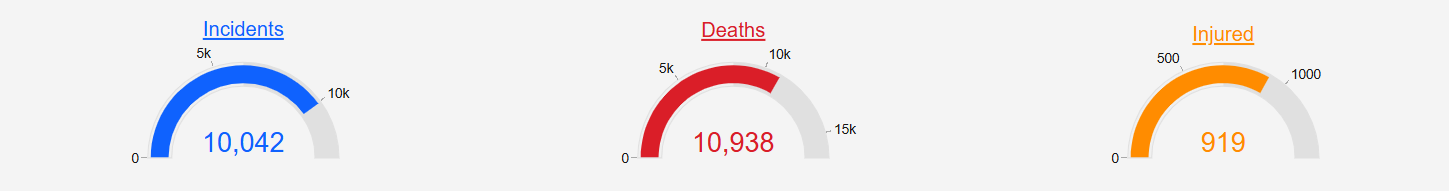
**6.a.3 Interactive Charts**

The Disaster Incident Dashboard is equipped with a range of interactive charts and visual components designed to provide multi-faceted insights into various incident types and their human impact across Bihar.

**1. Key Performance Indicator (KPI) Gauge Charts**

Prominently displayed at the top of the dashboard, these three gauge charts offer an immediate, high-level summary of critical incident metrics.

* **Total Incidents Gauge:** Displays the overall count of all reported incidents (e.g., drowning, road accidents, lightning) within the selected filters.
* **Total Deaths Gauge:** Shows the cumulative sum of all fatalities recorded across all incident types.
* **Total Injured Gauge:** Presents the total sum of all individuals reported as injured due to various incidents. These gauges feature distinct color coding (Blue for Incidents, Red for Deaths, Orange for Injured) and dynamically scale their maximum values based on the current data, providing a quick visual reference.



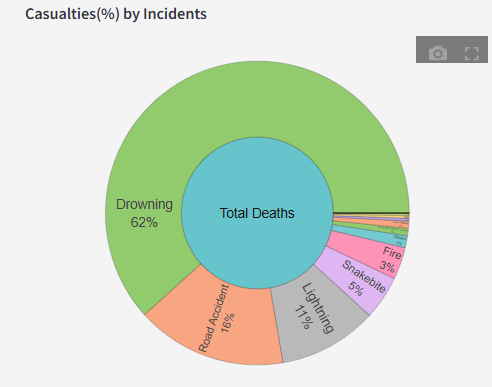
**2. Casualties Section (Interactive Incident Type Cards)**

Positioned strategically, this section presents interactive incident type cards that visually summarize death counts for various incident categories. Each colored card highlights a specific incident type (e.g., Drowning, Road Accident, Lightning, Fire) along with its associated death toll. These cards are dynamically sorted in descending order by death count, allowing for quick identification of the deadliest incident types. While the cards themselves are informative, they are static displays providing a quick summary rather than clickable filters.



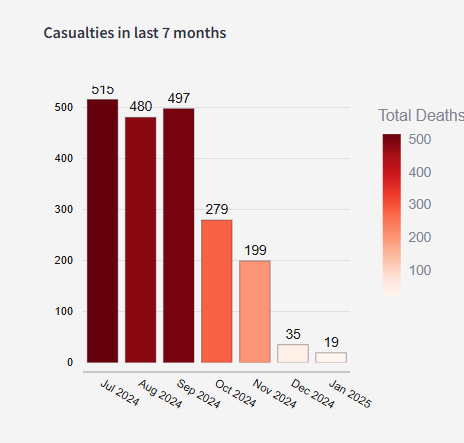
**3. Percentage of Deaths by Incident Type Chart**

This chart provides a critical percentage-based breakdown of fatalities by incident type. It visually represents the proportion of total deaths attributable to each specific category of incident (e.g., drowning, road accidents, lightning, fire). This visualization allows for quick identification of the most lethal incident types as a percentage of overall casualties, helping authorities to prioritize prevention strategies and resource allocation towards mitigating the most impactful threats. The chart updates dynamically based on applied filters, allowing for analysis of death percentages in specific districts or timeframes.



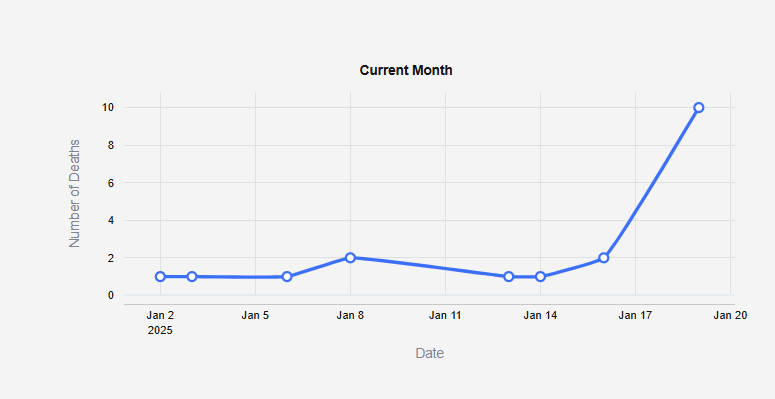
**4. Last 7 Months Deaths Chart**

This chart provides a focused temporal analysis, displaying the **number of deaths over the last 7 months** from a particular month that can be selected in the dashboard's filters. It also allows for granular analysis by showing death trends for **various incidents within a particular district** over the same seven-month period. This specific chart is crucial for reviewing recent incident patterns and casualties, supporting short-term response planning and trend observation.



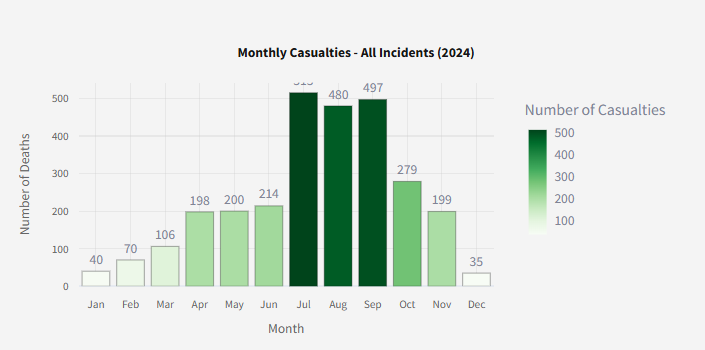
**5. Daily Deaths Chart**

This smooth spline line chart tracks daily death counts for the selected month and year. Designed for detailed temporal analysis, it displays a smooth curve with white circular markers indicating specific daily figures. Hover tooltips provide precise date and death count information. The chart's title typically indicates "Current Month," and it shows zero values for days without reported incidents, ensuring a continuous time series view. This helps in understanding daily fluctuations and immediate trends.



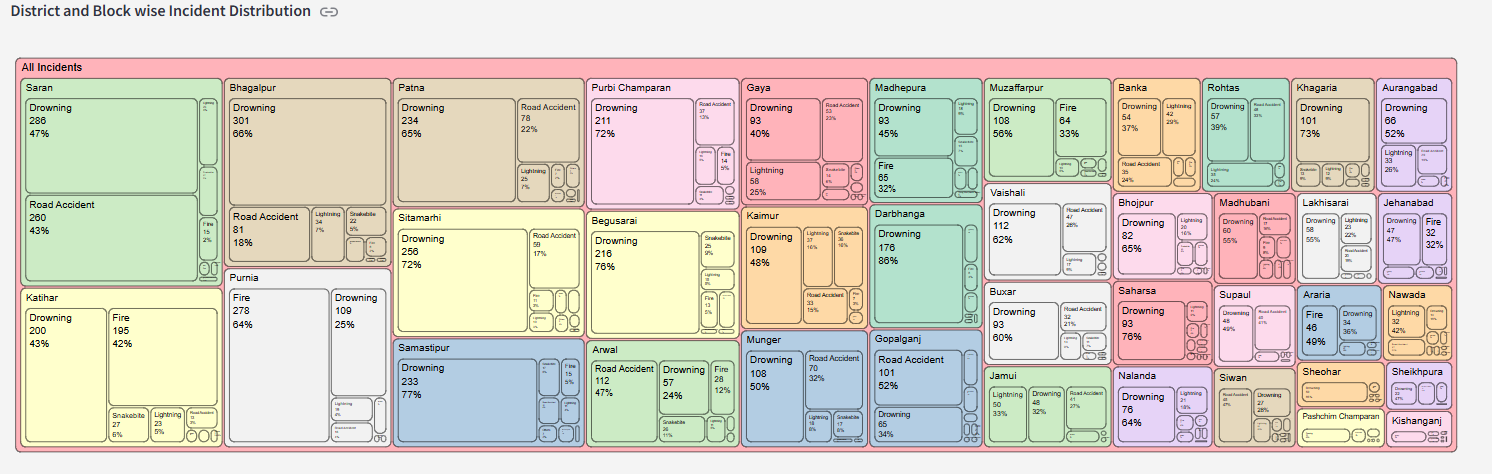
**6. Monthly Deaths Chart**

Presented as a column or bar chart with a green color gradient, this visualization aggregates monthly death counts for a selected year and incident type. It displays all 12 months (January to December), allowing users to discern seasonal patterns and monthly variations in casualties. Each bar includes a text label on top showing the exact death count. The chart's title dynamically updates based on the selected incident type and year, providing contextual clarity for seasonal pattern analysis.



**7. District and Block Treemap**

This powerful hierarchical treemap visualization is essential for understanding the geographical distribution of incidents. It organizes data by District, then further breaks it down by Incident Type within each district. The size of each rectangle within the treemap directly corresponds to the incident count, allowing for intuitive identification of areas and types with higher incident frequencies. Rectangles are color-coded based on incident count intensity. Users can hover over any part of the treemap to retrieve detailed information on the district, incident type, and incident count. This treemap offers a crucial hierarchical drill-down capability, enabling users to shift from a state-wide overview to granular block-level analysis within a selected district, which is vital for targeted emergency response and resource allocation.



## CHAPTER 7: ADVANTAGES

**7.a Bihar ColdWave Dashboard**

* **Centralized Information:** Provides a unified platform for all critical data pertaining to cold waves, consolidating fragmented information into a single source of truth.
* **Improved Situational Awareness:** Offers a rapid, visual, and comprehensive understanding of cold wave conditions across the state, enhancing the overall operational picture for authorities.
* **Data-Driven Decision Making:** Empowers government officials and emergency responders to make informed, strategic decisions based on real-time and historical data, leading to more effective and targeted interventions.
* **Efficient Resource Allocation:** Facilitates the identification of high-impact areas and critical needs, enabling precise prioritization and optimized deployment of emergency services, relief materials, and financial resources.
* **Enhanced Transparency:** Makes incident and relief operation data readily accessible to all relevant stakeholders, fostering greater accountability and public trust in government response efforts.
* **User-Friendly Interface:** The interactive and intuitive design of the dashboards ensures ease of use for all authorized personnel, requiring minimal technical expertise to navigate and extract valuable insights.
* **Proactive Planning:** Supports the development of more effective preparedness strategies by highlighting trends, vulnerable areas, and recurring incident patterns, moving towards more preventative measures.

**7.b Bihar Disaster Incident Dashboard**

* **Centralized Incident Data**: Provides a unified platform for all critical incident records, consolidating diverse reporting from various districts and blocks into a single source of truth for rapid access and analysis.
* **Enhanced Situational Awareness**: Offers a real-time, visual, and comprehensive understanding of various incident types, their locations, and associated human losses, significantly improving the operational picture for emergency responders and administrators.
* **Data-Driven Decision Making**: Empowers authorities to make informed, strategic decisions based on accurate incident data, enabling more effective and targeted interventions for public safety.
* **Optimized Resource Allocation**: Facilitates the identification of high-risk incident types and geographically vulnerable areas, allowing for precise prioritization and efficient deployment of emergency services, medical aid, and other essential resources.
* **Improved Accountability and Transparency**: Makes incident statistics and casualty data readily accessible to all relevant stakeholders, fostering greater accountability and public trust in government's incident response and reporting.
* **User-Friendly Interface**: The intuitive and interactive design ensures ease of use for all authorized personnel, allowing quick navigation and extraction of valuable insights without requiring extensive technical expertise.
* **Strategic Planning & Prevention**: Supports the development of more effective prevention strategies and public awareness campaigns by highlighting prevalent incident types, historical trends, and high-casualty areas, thereby minimizing future risks.

## CHAPTER 8: LIMITATIONS

**8.a Bihar ColdWave Dashboard**

* **Data Dependency and Accuracy**: The reliability and accuracy of the dashboard's insights are directly contingent upon the completeness, accuracy, and timeliness of the data entered into the underlying SQL Server database. Any inaccuracies or delays in raw data input will consequently affect the dashboard's displayed information.
* **Near Real-time Data Latency**: The dashboards provide near real-time data visualization. Data updates are governed by a defined refresh cycle (e.g., a 10-minute caching mechanism for the Cold Wave Dashboard, or a 30-minute TTL for the Incident Dashboard), meaning there might be a slight delay between an event occurring and its reflection on the dashboard.
* **Focus on Post-Incident Analysis**: The current scope of these projects is primarily focused on post-incident analysis and the monitoring of ongoing relief efforts. They do not currently incorporate advanced predictive analytics features for forecasting future events or their impacts.

**8.b Bihar Disaster Incident Dashboard**

* **Data Dependency and Accuracy:** The effectiveness and reliability of the dashboard's insights are directly contingent upon the completeness, accuracy, and timeliness of the incident data entered into the underlying SQL Server database. Inaccurate or delayed data inputs will inherently affect the dashboard's displayed information and subsequent analysis.
* **Near Real-time Data Latency:** The dashboard provides near real-time data visualization. Data updates are governed by a defined refresh cycle with an extended caching period (e.g., a 30-minute TTL or 1800 seconds), meaning there might be a slight delay between an incident occurring or being updated, and its reflection on the dashboard.
* **Focus on Post-Incident Analysis:** The current scope of the dashboard is primarily focused on post-incident analysis and the monitoring of ongoing incident impacts and casualties. It does not currently incorporate advanced predictive analytics features for forecasting future incidents or their potential effects.
* **Geographical Granularity:** While offering district and block-level analysis, the precision of geographical insights is dependent on the granularity of the input data and may not extend to micro-level mapping of incident sites.
* **Connectivity Requirements:** Consistent access to the dashboard relies on a stable network connection to ensure continuous data flow from the database and responsive user interaction.

# CHAPTER 9: CONCLUSION

The development of the Cold Wave Dashboard and the Disaster Incident Dashboard collectively represents a significant advancement in data-driven disaster and incident management for the Government of Bihar. By transforming complex, raw data into intuitive, interactive visualizations, KPIs, and analytical tools, these applications successfully provide clear and actionable overviews of both specific environmental hazards like cold waves and a diverse range of general incidents across the state. They effectively address the critical need for centralized monitoring systems, enabling rapid situational awareness, informed decision-making, and optimized resource allocation for the Emergency Operations Center and associated government bodies. Together, these dashboards serve as invaluable assets, substantially improving the speed, efficiency, and transparency of emergency response and public safety initiatives throughout Bihar, ultimately contributing to better preparedness and a more resilient state.

.

## CHAPTER 10: FUTURE SCOPE

The Cold Wave Dashboard and the Disaster Incident Dashboard serve as robust foundations for data-driven emergency management in Bihar. To further enhance their utility and impact, several key areas have been identified for future development and expansion:

* **Advanced Predictive Analytics:** Incorporating machine learning models to forecast potential cold wave severity, predict incident hotspots (e.g., accident-prone zones, areas vulnerable to lightning strikes), and anticipate resource needs, thereby enabling more proactive and preventative measures.
* **Mobile Application Development:** Creating dedicated mobile applications for field officials and ground-level responders. This would facilitate real-time data entry, instant access to critical information, and location-based alerts, significantly improving situational awareness and response efficiency in the field.
* **Enhanced Resource Management Modules:** Developing comprehensive modules to track the real-time movement and distribution of relief materials (e.g., blankets, medical supplies, food packets), deployment of personnel, and availability of infrastructure (e.g., night shelters, medical facilities). This would optimize logistics and accountability.
* **Integration with External Systems and Early Warning Systems:** Establishing seamless integration with other relevant government databases, weather forecasting services, remote sensing platforms, and early warning systems. This would enrich the dashboard's data, provide broader contextual insights, and facilitate automated alerts for impending conditions or emerging incidents.
* **Feedback and Collaboration Tools:** Implementing features for direct feedback from field units, enabling real-time communication and collaborative planning among different departments and agencies involved in disaster response.
* **Public Information Portal:** Potentially developing a public-facing component to share verified, actionable information with citizens, enhancing public safety and community resilience through awareness.

# CHAPTER 11: REFERENCES

Streamlit Documentation: <https://docs.streamlit.io/>

## Plotly Python Documentation: <https://plotly.com/python/>

* Pandas Documentation: <https://pandas.pydata.org/docs/>
* Bihar State Disaster Management Authority (BSDMA):

<https://bsdma.org/>

### CHAPTER 12: Full Source Code

**12.a. Bihar ColdWave Dashboard**

This section contains the complete Python source code for the main application file, Dashboard1.py.(Bihar ColdWave Dashboard)

import streamlit as st

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

from PIL import Image

import base64

from io import BytesIO

from datetime import datetime, timedelta

# Page Configuration handled by main.py(when we are merging all dashboards)

# Function to load and encode image to base64

def get\_image\_base64(path):

    try:

        img = Image.open(path)

        buffered = BytesIO()

        img\_format = path.split('.')[-1].upper()

        if img\_format == "PNG":

            img.save(buffered, format="PNG")

            mime\_type = "image/png"

        elif img\_format == "WEBP":

            img.save(buffered, format="WEBP")

            mime\_type = "image/webp"

        elif img\_format in ["JPG", "JPEG"]:

            img.save(buffered, format="JPEG")

            mime\_type = "image/jpeg"

        else:

            img.save(buffered, format="PNG")

            mime\_type = "image/png"

        img\_str = base64.b64encode(buffered.getvalue()).decode()

        return f"data:{mime\_type};base64,{img\_str}"

    except FileNotFoundError:

        st.warning(f"Image '{path}' not found. Displaying title without image or using default.")

        return None

    except Exception as e:

        print(f"Error encoding image {path}: {e}")

        return None

# Database connection function

@st.cache\_resource

def init\_db\_connection():

    """Establishes a SQLAlchemy engine connection to the SQL Server database using Dashboard1.toml config."""

    try:

        from sqlalchemy import create\_engine

        import os

        config\_path = os.path.join(".streamlit", "Dashboard1.toml")

        if os.path.exists(config\_path):

            # Creating SQLAlchemy connection string

            connection\_url = "mssql+pyodbc://KAKA/eoc?driver=ODBC+Driver+17+for+SQL+Server&trusted\_connection=yes&TrustServerCertificate=yes"

        else:

            # Fallback connection string

            connection\_url = "mssql+pyodbc://KAKA/eoc?driver=ODBC+Driver+17+for+SQL+Server&trusted\_connection=yes&TrustServerCertificate=yes"

        engine = create\_engine(connection\_url)

        return engine

    except Exception as e:

        st.error(f"Database connection failed. Check `Dashboard1.toml` and ensure DB is running. Error: {e}")

        return None

# Image to base64 conversion function

@st.cache\_data

def get\_image\_as\_base64(path):

    try:

        with open(path, "rb") as image\_file:

            encoded\_string = base64.b64encode(image\_file.read()).decode()

        image\_type = path.split('.')[-1].lower()

        if image\_type in ["jpg", "jpeg"]: return f"data:image/jpeg;base64,{encoded\_string}"

        elif image\_type == "png": return f"data:image/png;base64,{encoded\_string}"

        else: return f"data:image/png;base64,{encoded\_string}"

    except FileNotFoundError:

        st.warning(f"Image file not found at path: '{path}'. Please ensure it is in the correct directory.")

        return "data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7"

# Database-connected data loading function

@st.cache\_data(ttl=600)

def load\_data():

    try:

        sql\_query = """

        SELECT

            CD.RecordDate,

            CD.FYearID,

            D.DistrictName,

            B.BlockName,

            CD.AffectedPeople,

            CD.DeadPeople,

            CD.TotalNightShelter,

            PA\_Agg.TotalDistrictAllotedAmount AS AllotedAmount,

            CD.AmountSpent,

            CD.BlanketDistribution,

            CD.TotalPeopleNightShelter,

            CD.WoodWt,

            CD.BonfirePlace,

            D.DistrictCode

        FROM

            dbo.ColdWaveDetails AS CD

        LEFT JOIN

            (

                SELECT

                    DistrictCode,

                    SUM(AllotedAmount) AS TotalDistrictAllotedAmount

                FROM

                    dbo.ColdWavepaymentAllotment

                GROUP BY

                    DistrictCode

            ) AS PA\_Agg

            ON CD.DistrictCode = PA\_Agg.DistrictCode

        LEFT JOIN

            dbo.mst\_Districts AS D ON CD.DistrictCode = D.DistrictCode

        LEFT JOIN

            dbo.mst\_Blocks AS B ON CD.BlockCode = B.BlockCode AND CD.DistrictCode = B.DistrictCode;

        """

        engine = init\_db\_connection()

        if engine is None:

            return pd.DataFrame()

        df\_loaded = pd.read\_sql(sql\_query, engine)

        # SQLAlchemy engines handling connection cleanup automatically

        column\_mapping = {

            'RecordDate': 'date',

            'FYearID': 'financial\_year',

            'DistrictName': 'district',

            'BlockName': 'block',

            'AffectedPeople': 'affected\_population\_lac',

            'DeadPeople': 'death',

            'TotalNightShelter': 'rain\_basera',

            'AllotedAmount': 'alloted\_amount\_lac',

            'AmountSpent': 'expenditure\_amount\_lac',

            'BlanketDistribution': 'blanket\_distributed',

            'TotalPeopleNightShelter': 'people\_in\_rain\_basera',

            'WoodWt': 'wood\_burn\_kg',

            'BonfirePlace': 'bonfire\_places',

            'DistrictCode': 'district\_code'

        }

        df\_loaded.rename(columns=column\_mapping, inplace=True)

        if 'affected\_forms\_filled' not in df\_loaded.columns:

            df\_loaded['affected\_forms\_filled'] = 0

        df\_loaded['date'] = pd.to\_datetime(df\_loaded['date'], errors='coerce')

        numeric\_cols = [

            'affected\_forms\_filled', 'affected\_population\_lac', 'death', 'rain\_basera',

            'alloted\_amount\_lac', 'expenditure\_amount\_lac', 'blanket\_distributed',

            'people\_in\_rain\_basera', 'wood\_burn\_kg', 'bonfire\_places'

        ]

        for col in numeric\_cols:

            df\_loaded[col] = pd.to\_numeric(df\_loaded[col], errors='coerce').fillna(0)

        if 'district' in df\_loaded.columns:

            df\_loaded['district'] = df\_loaded['district'].astype(str).str.strip().str.title()

        if 'block' in df\_loaded.columns:

            df\_loaded['block'] = df\_loaded['block'].astype(str).str.strip().str.title()

        df\_loaded['district'] = df\_loaded['district'].fillna('Unknown')

        df\_loaded['block'] = df\_loaded['block'].fillna('Unknown')

        df\_loaded.dropna(subset=['date'], inplace=True)

        return df\_loaded

    except Exception as e:

        st.error(f"An error occurred while connecting to the database or loading data: {e}")

        st.error("Please check your database connection, secrets file, and SQL query.")

        return pd.DataFrame()

def run():

    # --- Load data ---

    df\_main = load\_data()

    if df\_main.empty:

        st.error("No data available. Please check your database connection.")

        return

    # Preparing filter data for horizontal layout below header

    if 'district' in df\_main.columns and not df\_main['district'].empty:

        unique\_districts = sorted(list(df\_main['district'].unique()))

    else:

        unique\_districts = []

    min\_date\_data = pd.to\_datetime('2022-12-15').date()

    if not df\_main.empty and 'date' in df\_main.columns and not df\_main['date'].min() is pd.NaT:

        min\_date\_data = min(min\_date\_data, df\_main['date'].min().date())

    # Setting the default end date for the filter to today's actual date

    default\_end\_date\_filter = datetime.now().date()

    # The default start date for the filter should be the min date of our records

    default\_start\_date\_filter = min\_date\_data

    # Ensuring the default range is valid (start <= end)

    if default\_start\_date\_filter > default\_end\_date\_filter:

        default\_start\_date\_filter = default\_end\_date\_filter - timedelta(days=30) # Fallback to 30 days before if somehow start > end

    # Main Page Header

    eoc\_logo\_header\_base64 = get\_image\_as\_base64("eoc\_logo.png")

    header\_logo\_html = f'<img src="{eoc\_logo\_header\_base64}" alt="EOC" class="header-logo-img">' if eoc\_logo\_header\_base64 else ""

    st.markdown(f"""

        <div class="dashboard-header">

            {header\_logo\_html}

            <div class="header-title-block">

                <h1>ColdWave Dashboard</h1>

                <p class="tagline">Emergency Operations Center - Government of Bihar</p>

            </div>

        </div>

    """, unsafe\_allow\_html=True)

    # Creating minimal horizontal layout for filters

    col1, col2, col3 = st.columns([1, 1, 1.2])

    with col1:

        selected\_district\_filter = st.selectbox(

            "District",

            options=unique\_districts,

            key="sb\_dist\_filter",

            placeholder="Select District",

            index=None

        )

    with col2:

        if 'block' in df\_main.columns:

            if selected\_district\_filter:

                unique\_blocks\_for\_district = df\_main[df\_main['district'] == selected\_district\_filter]['block'].unique()

                block\_options = sorted(list(unique\_blocks\_for\_district))

            else:

                block\_options = sorted(list(df\_main['block'].unique()))

        else:

            block\_options = []

        selected\_block\_filter = st.selectbox(

            "Block",

            options=block\_options,

            key="sb\_block\_filter",

            placeholder="Select Block",

            index=None

        )

    with col3:

        date\_input\_value = st.date\_input(

            "Select Date Range",

            value=[default\_start\_date\_filter, default\_end\_date\_filter],

            min\_value=min\_date\_data,

            max\_value=default\_end\_date\_filter,

            key="di\_date"

        )

    # Processing date range

    if len(date\_input\_value) == 2:

        start\_date\_filter\_selected, end\_date\_filter\_selected = pd.to\_datetime(date\_input\_value[0]), pd.to\_datetime(date\_input\_value[1])

        if start\_date\_filter\_selected > end\_date\_filter\_selected:

            st.error("Error: Start date cannot be after end date. Adjusting range to default.")

            start\_date\_filter\_selected, end\_date\_filter\_selected = pd.to\_datetime(default\_start\_date\_filter), pd.to\_datetime(default\_end\_date\_filter)

        date\_range = [start\_date\_filter\_selected, end\_date\_filter\_selected]

    else: # This handles the case where only one date is selected, which can happen initially

        date\_range = [pd.to\_datetime(default\_start\_date\_filter), pd.to\_datetime(default\_end\_date\_filter)]

    st.markdown("""

<style>

/\* Keyframes for Single Line Border Tracing Animation \*/

@keyframes border-trace-single {

    0% {

        box-shadow: 0 0 0 0px #0043CE;

    }

    100% {

        box-shadow: 0 0 0 2px #0043CE; /\* Blue border effect \*/

    }

}

section.main {

    transition: margin-left 0.3s ease-out !important;

}

body, .stApp { font-family: "IBM Plex Sans", sans-serif !important; font-size: 0.95rem; background-color: #F4F4F4; }

h1, h3, h6 { font-family: "IBM Plex Sans", sans-serif; }

h3 { font-size: 1.1em !important; margin-bottom: 5px !important; margin-top: 10px !important; font-weight: 600; color: #161616; }

h6 { font-size: 0.85em !important; color: #525252; text-align: left; font-weight: 600; margin-bottom: 5px; }

/\* Adjusted margin for the horizontal rule to be more compact \*/

hr { margin: 5px 0px !important; }

/\* Dashboard Header Styling \*/

.dashboard-header {

    background-color: #004C99; color: #FFFFFF;

    padding: 0.3375rem 1.0125rem; text-align: left; display: flex;

    align-items: center; border-bottom: 2.025px solid #ffc107;

    border-radius: 5.4px;

    box-shadow: 0 1.35px 2.7px rgba(0,0,0,0.1);

    margin-bottom: 0.675rem;

}

.dashboard-header .header-logo-img { height: 20.25px; margin-right: 10.125px; }

.dashboard-header .header-title-block { flex-grow: 1; }

.dashboard-header h1 {

    font-size: 1.0125rem; font-weight: 600; margin: 0 0 0.0675rem 0;

    line-height: 0.81; color: #FFFFFF;

}

.dashboard-header p.tagline { font-size: 0.54rem; margin: 0; color: #e0e0e0; opacity: 0.9; }

.kpi-card-single {

    background-color: #FFFFFF;

    border: 1px solid #00BCD4; /\* Thin, fine blue (cyan) boundary \*/

    border-radius: 8px; /\* Applied rounded corners here \*/

    box-shadow: 0 1px 3px rgba(0, 0, 0, 0.05);

    transition: transform 0.2s ease-out, box-shadow 0.2s ease-out, border 0.2s ease-out; /\* Add border to transition \*/

    max-width: 95%;

    margin-left: auto;

    margin-right: auto;

    margin-bottom: 10px;

    position: relative;

    display: flex;

    flex-direction: column;

    height: 85px;

    overflow: hidden;

}

.kpi-card-single::before {

    content: '';

    position: absolute;

    top: 0;

    left: 0;

    right: 0;

    bottom: 0;

    background: linear-gradient(90deg, hsla(211, 100%, 91%, 1) 2%, hsla(275, 45%, 78%, 1) 100%);

    opacity: 0;

    transition: opacity 0.3s ease-in-out;

    z-index: 1;

    pointer-events: none;

    border-radius: 6px; /\* Ensure gradient also respects rounded corners \*/

}

.kpi-card-single:hover::before {

    opacity: 1;

}

.kpi-card-single:hover {

    transform: translateY(-5px) scale(1.03);

    /\* Updated: Lighter blue border (#66B3FF) and thinner (0.5px) \*/

    box-shadow: 0 6px 15px rgba(0, 0, 0, 0.1), 0 0 0 0.5px #66B3FF;

}

.kpi-card-single .kpi-title,

.kpi-card-single .kpi-values-container,

.kpi-card-single .kpi-value-block,

.kpi-card-single .kpi-sublabel,

.kpi-card-single .kpi-value-main {

    position: relative;

    z-index: 2;

}

.kpi-card-single .kpi-title {

    font-size: 0.85em;

    color: #393939;

    text-transform: uppercase;

    letter-spacing: 0.3px;

    font-weight: normal;

    margin-bottom: 1px;

    padding-bottom: 2px;

    border-bottom: 1px solid #F0F0F0;

    text-align: center;

    white-space: normal;

    /\* Adjusted height and line-height for vertical centering of single-line titles \*/

    height: 2.2em; /\* Reduced height slightly to bring it down \*/

    line-height: 2.2em; /\* Set line-height equal to height for vertical centering \*/

}

/\* Specific style for the multi-line "Affected/ Form Filled" title to maintain its appearance \*/

.kpi-card-single .kpi-title:has(.kpi-values-container + .kpi-values-container) {

    height: auto; /\* Reset height for merged cards \*/

    line-height: normal; /\* Reset line-height for merged cards \*/

}

/\* Ensuring the "Affected/ Form Filled Blocks & Nagar Nikaay" title retains its original multi-line behavior \*/

.kpi-card-single .kpi-title:contains("AFFECTED/ FORM FILLED BLOCKS & NAGAR NIKAAY") {

    height: 2.6em; /\* Original height for this specific title \*/

    line-height: 1.3; /\* Original line-height for this specific title \*/

}

.kpi-card-single .kpi-values-container {

    display: flex;

    justify-content: space-around;

    align-items: center;

    width: 100%;

    flex-grow: 1;

    margin-top: 2px;

}

.kpi-card-single .kpi-value-block {

    text-align: center;

    padding: 0 2px;

}

.kpi-card-single .kpi-sublabel {

    font-size: 0.55em;

    color: #4A4A4A;

    margin-bottom: 0px;

    display: block;

    font-weight: 400;

    transition: font-weight 0.2s ease-in-out; /\* Add transition for smooth bolding \*/

}

/\* Making KPI sublabels bold on hover \*/

.kpi-card-single:hover .kpi-sublabel {

    font-weight: bold !important;

}

.kpi-card-single .kpi-value-main {

    font-size: 1.05em;

    font-weight: 600;

    color: #0F62FE;

    line-height: 1.1;

    display: block;

}

.kpi-card-merged {

    background-color: #FFFFFF;

    height: 170px;

}

.kpi-card-merged .merged-kpi-row {

    flex-grow: 1;

}

.kpi-card-merged .kpi-title {

    border-bottom: none;

    font-size: 0.65em;

    text-align: center;

    margin-bottom: 0;

    padding-bottom: 0;

    height: auto;

    line-height: normal;

}

.kpi-card-merged:hover {

    transform: translateY(-5px) scale(1.03);

    box-shadow: 0 6px 15px rgba(0, 0, 0, 0.1), 0 0 0 0.5px #66B3FF;

    position: relative;

}

div[data-testid="stSelectbox"] div[data-baseweb="select"] > div,

div[data-testid="stDateInput"] > div > div {

    transition: all 0.2s ease-in-out;

    border: 1px solid transparent;

    border-radius: 4px;

}

div[data-testid="stSelectbox"]:hover div[data-baseweb="select"] > div,

div[data-testid="stDateInput"]:hover > div > div {

    border: 1px solid #0F62FE;

    box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

.footer-card { background-color: #343a40; color: #dee2e6; padding: 1rem; border-radius: 8px; margin-top: 1.5rem; text-align: center; box-shadow: 0 -1px 3px rgba(0,0,0,0.05); border-top: 1px solid var(--card-border-color); font-size: 0.8rem; box-sizing: border-box; }

.footer-title { color: #ffc107 !important; font-size: 1rem !important; font-weight: 500 !important; margin-bottom: 0.5rem !important; }

.footer-card p { margin-bottom: 0.25rem; font-size: 0.75rem;}

</style>

    """, unsafe\_allow\_html=True)

    if df\_main.empty:

        st.error("Dashboard cannot be displayed because no data could be loaded.")

        st.stop()

    # Applying filters to create `kpi\_ts\_df` for charts and "Till Now" KPIs

    # This dataframe is based on the selected date range.

    base\_filtered\_df = df\_main.copy()

    start\_date\_filtered, end\_date\_filtered = date\_range[0], date\_range[1]

    base\_filtered\_df = base\_filtered\_df[(base\_filtered\_df['date'] >= start\_date\_filtered) & (base\_filtered\_df['date'] <= end\_date\_filtered)]

    kpi\_ts\_df = base\_filtered\_df.copy() # This will be used for Time Series charts and 'Till Now' KPIs

    if selected\_district\_filter and 'district' in kpi\_ts\_df.columns:

        kpi\_ts\_df = kpi\_ts\_df[kpi\_ts\_df['district'] == selected\_district\_filter]

        if selected\_block\_filter and 'block' in kpi\_ts\_df.columns:

            kpi\_ts\_df = kpi\_ts\_df[kpi\_ts\_df['block'] == selected\_block\_filter]

    today\_kpi\_reference\_date = end\_date\_filter\_selected

    today\_df = kpi\_ts\_df[kpi\_ts\_df['date'] == today\_kpi\_reference\_date.normalize()]

    tillnow\_df\_for\_kpis = kpi\_ts\_df.copy()

    if kpi\_ts\_df.empty:

        ts\_df = pd.DataFrame(columns=['date', 'affected\_forms\_filled', 'affected\_population\_lac', 'death', 'rain\_basera', 'alloted\_amount\_lac', 'expenditure\_amount\_lac', 'blanket\_distributed', 'people\_in\_rain\_basera', 'wood\_burn\_kg', 'bonfire\_places'])

        ts\_df['date'] = pd.to\_datetime(ts\_df['date'])

    else:

        # Aggregation for time series data

        ts\_df = kpi\_ts\_df.groupby('date', as\_index=False).agg({

            'affected\_forms\_filled': 'sum',

            'affected\_population\_lac': 'sum',

            'death': 'sum',

            'rain\_basera': 'sum',

            'expenditure\_amount\_lac': 'sum',

            'blanket\_distributed': 'sum',

            'people\_in\_rain\_basera': 'sum',

            'wood\_burn\_kg': 'sum',

            'bonfire\_places': 'sum'

        })

        if 'alloted\_amount\_lac' in kpi\_ts\_df.columns and 'district' in kpi\_ts\_df.columns:

            alloted\_amount\_ts = kpi\_ts\_df.groupby(['date', 'district'])['alloted\_amount\_lac'].max().reset\_index()

            alloted\_amount\_ts\_daily\_sum = alloted\_amount\_ts.groupby('date')['alloted\_amount\_lac'].sum().reset\_index()

            ts\_df = pd.merge(ts\_df, alloted\_amount\_ts\_daily\_sum, on='date', how='left')

            ts\_df['alloted\_amount\_lac'] = ts\_df['alloted\_amount\_lac'].fillna(0)

        else:

            ts\_df['alloted\_amount\_lac'] = 0

    plotly\_template="plotly\_white"

    primary\_color="#0F62FE"

    secondary\_color="#525252"

    death\_color="#DA1E28"

    blanket\_color="#24A148"

    exp\_color="#A8A8A8"

    fill\_opacity=0.1

    text\_color="#161616"

    grid\_color="#E0E0E0"

    axis\_color="#C6C6C6"

    font\_family="IBM Plex Sans, sans-serif"

    theme\_secondary\_bg\_color="#FFFFFF"

    def style\_chart(fig, chart\_height=190, is\_pie\_or\_donut=False):

        fig.update\_layout(

            template=plotly\_template,

            plot\_bgcolor='rgba(0,0,0,0)',

            paper\_bgcolor='rgba(0,0,0,0)',

            xaxis=dict(showgrid=False, linecolor=axis\_color, tickfont=dict(color=text\_color, size=9), showline=True, zeroline=False),

            yaxis=dict(showgrid=True, gridcolor=grid\_color, linecolor=axis\_color, tickfont=dict(color=text\_color, size=9), showline=True, zeroline=False),

            hoverlabel=dict(bgcolor="#FFFFFF", font\_size=11, font\_family=font\_family, bordercolor=axis\_color, font\_color=text\_color),

            legend=dict(orientation="h", yanchor="bottom", y=1.05, xanchor="right", x=1, font=dict(color=text\_color, size=9), bgcolor='rgba(0,0,0,0)'),

            margin=dict(l=30, r=10, t=10, b=20),

            title\_text='',

            title\_x=0.5,

            font\_color=text\_color,

            font\_family=font\_family,

            height=chart\_height

        )

        if not is\_pie\_or\_donut:

            fig.update\_layout(hovermode='x unified')

            fig.update\_xaxes(title\_text=None, hoverformat='%a, %d %b %Y')

            fig.update\_yaxes(title\_text=None)

        else:

            fig.update\_layout(xaxis=dict(visible=False, showgrid=False), yaxis=dict(visible=False, showgrid=False))

        return fig

    if kpi\_ts\_df.empty and not df\_main.empty:

        st.warning("No data available for KPIs and Time Series charts for the selected filter combination. Please adjust filters.")

    def create\_single\_kpi\_card(title, today\_val, till\_now\_val, is\_lac=False):

        today\_str = f"{today\_val:,.2f}" if is\_lac else f"{int(today\_val):,}"

        till\_now\_str = f"{till\_now\_val:,.2f}" if is\_lac else f"{int(till\_now\_val):,}"

        st.markdown(f"""

            <div class="kpi-card-single">

                <div class="kpi-title">{title}</div>

                <div class="kpi-values-container">

                    <div class="kpi-value-block">

                        <span class="kpi-sublabel">Today</span>

                        <span class="kpi-value-main">{today\_str}</span>

                    </div>

                    <div class="kpi-value-block">

                        <span class="kpi-sublabel">Till Now</span>

                        <span class="kpi-value-main">{till\_now\_str}</span>

                    </div>

                </div>

            </div>

        """, unsafe\_allow\_html=True)

    def create\_merged\_financial\_card(allot\_today, allot\_till, exp\_today, exp\_till):

        allot\_today\_str = f"{allot\_today:,.2f}"

        allot\_till\_str = f"{allot\_till:,.2f}"

        exp\_today\_str = f"{exp\_today:,.2f}"

        exp\_till\_str = f"{exp\_till:,.2f}"

        st.markdown(f"""

            <div class="kpi-card-single kpi-card-merged">

                <div class="merged-kpi-row" style="flex-grow: 1; border-bottom: 1px solid #F0F0F0; margin-bottom: 5px; padding-bottom: 5px;">

                    <div class="kpi-title">ALLOTED AMOUNT (LAC)</div>

                    <div class="kpi-values-container">

                        <div class="kpi-value-block">

                            <span class="kpi-sublabel">Today</span>

                            <span class="kpi-value-main">{allot\_today\_str}</span>

                        </div>

                        <div class="kpi-value-block">

                            <span class="kpi-sublabel">Till Now</span>

                            <span class="kpi-value-main">{allot\_till\_str}</span>

                        </div>

                    </div>

                </div>

                <div class="merged-kpi-row" style="flex-grow: 1;">

                    <div class="kpi-title">EXPENDITURE AMOUNT (LAC)</div>

                    <div class="kpi-values-container">

                        <div class="kpi-value-block">

                            <span class="kpi-sublabel">Today</span>

                            <span class="kpi-value-main">{exp\_today\_str}</span>

                        </div>

                        <div class="kpi-value-block">

                            <span class="kpi-sublabel">Till Now</span>

                            <span class="kpi-value-main">{exp\_till\_str}</span>

                        </div>

                    </div>

                </div>

            </div>

        """, unsafe\_allow\_html=True)

    st.markdown("---")

    main\_col1, main\_col2 = st.columns([0.55, 0.45], gap="large")

    with main\_col1:

        chart\_row1\_col1, chart\_row1\_col2 = st.columns(2, gap="medium")

        with chart\_row1\_col1:

            st.markdown("###### Affected Population (lac)")

            if not ts\_df.empty and 'affected\_population\_lac' in ts\_df.columns:

                fig = px.line(ts\_df, x='date', y='affected\_population\_lac', color\_discrete\_sequence=[primary\_color])

                fig.update\_traces(mode='lines', line\_shape='linear', line=dict(width=1.5), fill='tozeroy', fillcolor=f'rgba(15, 98, 254, {fill\_opacity})', name='Population', hovertemplate='%{y:,.2f} lac<extra></extra>')

                st.plotly\_chart(style\_chart(fig), use\_container\_width=True)

            else:

                st.caption("No data for Population.")

        with chart\_row1\_col2:

            st.markdown("###### Deaths Reported")

            if not ts\_df.empty and 'death' in ts\_df.columns:

                fig = px.bar(ts\_df, x='date', y='death', color\_discrete\_sequence=[death\_color])

                fig.update\_traces(name='Deaths', hovertemplate='%{y:,}<extra></extra>', marker\_line\_width=0)

                fig.update\_layout(bargap=0.6)

                st.plotly\_chart(style\_chart(fig), use\_container\_width=True)

            else:

                st.caption("No data for Deaths.")

        chart\_row2\_col1, chart\_row2\_col2 = st.columns(2, gap="medium")

        with chart\_row2\_col1:

            st.markdown("###### Shelter & Blankets")

            if not ts\_df.empty and 'people\_in\_rain\_basera' in ts\_df.columns and 'blanket\_distributed' in ts\_df.columns:

                fig = go.Figure()

                fig.add\_trace(go.Scatter(x=ts\_df['date'], y=ts\_df['people\_in\_rain\_basera'], name='People Sheltered', mode='lines', line=dict(color=secondary\_color, width=1.5), fill='tozeroy', fillcolor=f'rgba(82, 82, 82, {fill\_opacity})', hovertemplate='Sheltered: %{y:,}<extra></extra>'))

                fig.add\_trace(go.Scatter(x=ts\_df['date'], y=ts\_df['blanket\_distributed'], name='Blankets Distributed', mode='lines', line=dict(color=blanket\_color, width=1.5, dash='dot'), yaxis='y2', hovertemplate='Blankets: %{y:,}<extra></extra>'))

                fig = style\_chart(fig)

                fig.update\_layout(yaxis=dict(tickfont=dict(color=secondary\_color)), yaxis2=dict(title=None, overlaying='y', side='right', showgrid=False, showline=True, linecolor=axis\_color, tickfont=dict(color=blanket\_color)), legend=dict(y=1.15))

                st.plotly\_chart(fig, use\_container\_width=True)

            else:

                st.caption("No data for Shelter & Blankets.")

        with chart\_row2\_col2:

            st.markdown("###### Financial Overview (Lac)")

            if not ts\_df.empty and 'alloted\_amount\_lac' in ts\_df.columns and 'expenditure\_amount\_lac' in ts\_df.columns:

                fig = px.line(ts\_df, x='date', y=['alloted\_amount\_lac', 'expenditure\_amount\_lac'], color\_discrete\_sequence=[primary\_color, exp\_color])

                fig.update\_traces(mode='lines', line\_shape='linear', line=dict(width=1.5), hovertemplate='%{y:,.2f} lac<extra></extra>')

                fig = style\_chart(fig)

                if len(fig.data) >= 1:

                    fig.data[0].name = 'Alloted'

                if len(fig.data) >= 2:

                    fig.data[1].name = 'Expenditure'

                st.plotly\_chart(fig, use\_container\_width=True)

            else:

                st.caption("No data for Financial Overview.")

    with main\_col2:

        st.markdown("###### District/Block Overview")

        treemap\_input\_data = base\_filtered\_df.copy()

        if selected\_district\_filter:

            treemap\_input\_data = treemap\_input\_data[treemap\_input\_data['district'] == selected\_district\_filter]

        if selected\_district\_filter:

            if not treemap\_input\_data.empty:

                block\_level\_data = treemap\_input\_data.groupby(['district', 'block'], as\_index=False)['affected\_population\_lac'].sum()

                treemap\_path = [px.Constant("Filtered Overview"), 'district', 'block']

                caption\_text = "No data for Treemap for the selected block."

            else:

                block\_level\_data = pd.DataFrame()

                caption\_text = "No data for Treemap for the selected district."

        else:

            if not treemap\_input\_data.empty:

                block\_level\_data = treemap\_input\_data.groupby(['district'], as\_index=False)['affected\_population\_lac'].sum()

                treemap\_path = [px.Constant("Filtered Overview"), 'district']

                caption\_text = "No data for Treemap."

            else:

                block\_level\_data = pd.DataFrame()

                caption\_text = "No data for Treemap."

        if not block\_level\_data.empty:

            block\_level\_data['affected\_population\_lac'] = pd.to\_numeric(block\_level\_data['affected\_population\_lac'], errors='coerce').fillna(0)

            block\_level\_data = block\_level\_data[block\_level\_data['affected\_population\_lac'] >= 0]

        if not block\_level\_data.empty:

            fig\_treemap = px.treemap(

                block\_level\_data,

                path=treemap\_path,

                values='affected\_population\_lac',

                color='affected\_population\_lac',

                custom\_data=['affected\_population\_lac'],

                color\_continuous\_scale='Blues',

                title=None

            )

            fig\_treemap.update\_layout(

                height=500,

                margin=dict(t=10, l=10, r=10, b=10),

                font\_family=font\_family,

                hoverlabel=dict(bgcolor="#FFFFFF", font\_size=11, font\_family=font\_family, bordercolor=axis\_color, font\_color=text\_color),

                coloraxis\_colorbar=dict(title="Pop. (Lac)")

            )

            fig\_treemap.update\_traces(

                texttemplate="<b>%{label}</b><br>%{value:,.2f}",

                hovertemplate='<b>%{label}</b><br>Population (Lac): %{value:,.2f} L<extra></extra>',

                textposition='middle center',

                textfont\_size=11,

                marker=dict(cornerradius=0, line=dict(color='#B0B0B0', width=0.5), pad=dict(t=2,l=2,r=2,b=2))

            )

            fig\_treemap.update\_traces(maxdepth=len(treemap\_path))

            st.plotly\_chart(fig\_treemap, use\_container\_width=True)

        else:

            st.caption(caption\_text)

    total\_allotment\_dashboard = 0

    if 'alloted\_amount\_lac' in tillnow\_df\_for\_kpis.columns and 'district' in tillnow\_df\_for\_kpis.columns and not tillnow\_df\_for\_kpis.empty:

        unique\_district\_allotments = tillnow\_df\_for\_kpis.groupby('district')['alloted\_amount\_lac'].max()

        total\_allotment\_dashboard = unique\_district\_allotments.sum()

    else:

        total\_allotment\_dashboard = 0

    total\_expenditure\_dashboard = tillnow\_df\_for\_kpis['expenditure\_amount\_lac'].sum() if not tillnow\_df\_for\_kpis.empty else 0

    # For "Today's" Allotment KPI:

    today\_allotment = 0

    if not today\_df.empty and 'district' in today\_df.columns and 'alloted\_amount\_lac' in today\_df.columns:

        today\_allotment = today\_df.groupby('district')['alloted\_amount\_lac'].max().sum()

    today\_expenditure = today\_df['expenditure\_amount\_lac'].sum() if not today\_df.empty else 0

    kpi\_data = {

        "forms": {"title": "Affected/ Form Filled Blocks & Nagar Nikaay", "today": today\_df['affected\_forms\_filled'].sum(), "till\_now": tillnow\_df\_for\_kpis['affected\_forms\_filled'].sum(), "is\_lac": False},

        "population": {"title": "AFFECTED POPULATION", "today": today\_df['affected\_population\_lac'].sum(), "till\_now": tillnow\_df\_for\_kpis['affected\_population\_lac'].sum(), "is\_lac": True},

        "basera": {"title": "NO. OF RAIN BASERA", "today": today\_df['rain\_basera'].sum(), "till\_now": tillnow\_df\_for\_kpis['rain\_basera'].sum(), "is\_lac": False},

        "deaths": {"title": "NO OF DEATHS", "today": today\_df['death'].sum(), "till\_now": tillnow\_df\_for\_kpis['death'].sum(), "is\_lac": False},

        "people\_basera": {"title": "NO. OF PEOPLE IN RAIN BASERA", "today": today\_df['people\_in\_rain\_basera'].sum(), "till\_now": tillnow\_df\_for\_kpis['people\_in\_rain\_basera'].sum(), "is\_lac": False},

        "blankets": {"title": "BLANKETS DISTRIBUTED", "today": today\_df['blanket\_distributed'].sum(), "till\_now": tillnow\_df\_for\_kpis['blanket\_distributed'].sum(), "is\_lac": False},

        "wood": {"title": "TOTAL WOOD BURN (IN KG)", "today": today\_df['wood\_burn\_kg'].sum(), "till\_now": tillnow\_df\_for\_kpis['wood\_burn\_kg'].sum(), "is\_lac": False},

        "bonfires": {"title": "NO. OF BONFIRE PLACES", "today": today\_df['bonfire\_places'].sum(), "till\_now": tillnow\_df\_for\_kpis['bonfire\_places'].sum(), "is\_lac": False},

        "allotment": {"today": today\_allotment, "till\_now": total\_allotment\_dashboard},

        "expenditure": {"today": today\_expenditure, "till\_now": total\_expenditure\_dashboard},

    }

    st.markdown("---")

    kpi\_order = ["forms", "population", "basera", "financial", "deaths", "people\_basera", "blankets", "wood", "bonfires"]

    cols = st.columns(3)

    col\_idx = 0

    for key in kpi\_order:

        current\_col = cols[col\_idx]

        with current\_col:

            if key == "financial":

                create\_merged\_financial\_card(allot\_today=kpi\_data["allotment"]["today"], allot\_till=kpi\_data["allotment"]["till\_now"], exp\_today=kpi\_data["expenditure"]["today"], exp\_till=kpi\_data["expenditure"]["till\_now"])

            else:

                c = kpi\_data[key]

                create\_single\_kpi\_card(c["title"], c["today"], c["till\_now"], c["is\_lac"])

        col\_idx = (col\_idx + 1) % 3

    # Footer

    st.markdown(f'''<div class="footer-card" style="margin-top: 2rem;">

        <h4 class="footer-title">🚨 Bihar Disaster Management Dashboard</h4>

        <p>Built with Streamlit & Plotly | Last updated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</p>

        <p>Data visualization for disaster preparedness and emergency response planning.</p>

        <p style="font-size: 0.7rem; margin-top: 0.4rem;">🏛️ Government of Bihar | Emergency Operations Center</p>

        </div>''', unsafe\_allow\_html=True)

if \_\_name\_\_ == "\_\_main\_\_":

    run()

**12.b. Bihar Disaster Incident Dashboard**

This section contains the complete Python source code for the main application file, Dashboard2.py.(Bihar Disaster Incident Dashboard)

import streamlit as st

import pandas as pd

import numpy as np

import plotly.express as px

import plotly.graph\_objects as go

from PIL import Image

import base64

from io import BytesIO

from datetime import datetime, timedelta

import time

def run():

    import pandas as pd

    from datetime import datetime, timedelta

    try:

        from streamlit\_plotly\_events import plotly\_events

        PLOTLY\_EVENTS\_AVAILABLE = True

    except ImportError:

        PLOTLY\_EVENTS\_AVAILABLE = False

    # Page Configuration is being handled by main.py

    st.markdown("""

<style>

    @keyframes fadeIn {

        0% { opacity: 0; transform: translateY(15px) scale(0.98); }

        100% { opacity: 1; transform: translateY(0) scale(1); }

    }

    body, .stApp {

        font-family: "IBM Plex Sans", sans-serif !important;

        font-size: 0.95rem;

        background-color: #F4F4F4;

    }

    .main .block-container {

        padding-top: 0.5rem !important;

        padding-bottom: 0.5rem !important;

        max-width: 100% !important;

    }

    .stColumn > div {

        padding: 0.2rem !important;

    }

    .element-container {

        margin-bottom: 0.3rem !important;

    }

    .stMarkdown {

        margin-bottom: 0.2rem !important;

    }

    div[data-testid="stPlotlyChart"] {

        margin-bottom: 0.5rem !important;

    }

    .dashboard-header {

        background-color: #004C99; color: #FFFFFF;

        padding: 0.3375rem 1.0125rem; text-align: left; display: flex;

        align-items: center; border-bottom: 2.025px solid #ffc107;

        border-radius: 5.4px;

        box-shadow: 0 1.35px 2.7px rgba(0,0,0,0.1);

        margin-bottom: 0.675rem;

        margin-left: -1rem;

        margin-right: -1rem;

        margin-top: -1rem;

    }

    .dashboard-header .header-logo-img { height: 20.25px; margin-right: 10.125px; }

    .dashboard-header .header-title-block { flex-grow: 1; }

    .dashboard-header h1 {

        font-size: 1.0125rem; font-weight: 600; margin: 0 0 0.0675rem 0;

        line-height: 0.81; color: #FFFFFF;

    }

    .dashboard-header p.tagline { font-size: 0.54rem; margin: 0; color: #e0e0e0; opacity: 0.9; }

    .dashboard-divider-full {

        border: none !important;

        height: 2px !important;

        background: linear-gradient(90deg, #0F62FE 0%, #4589FF 100%) !important;

        margin-top: 8px !important;

        margin-bottom: 16px !important;

        width: 100% !important;

        margin-left: -1rem !important;

        margin-right: -1rem !important;

        border-radius: 2px !important;

        box-shadow: 0 2px 4px rgba(15, 98, 254, 0.3) !important;

        z-index: 10 !important;

        display: block !important;

    }

    .dashboard-divider {

        border: none !important;

        height: 2px !important;

        background: linear-gradient(90deg, #0F62FE 0%, #4589FF 100%) !important;

        margin-top: 12px !important;

        margin-bottom: 16px !important;

        width: 100% !important;

        border-radius: 1px !important;

        box-shadow: 0 1px 3px rgba(15, 98, 254, 0.2) !important;

        display: block !important;

    }

    hr.dashboard-divider-full,

    hr.dashboard-divider {

        opacity: 1 !important;

        visibility: visible !important;

    }

    div[data-testid="stPlotlyChart"]:has(.js-plotly-plot[data-unformatted\*="Daily Deaths"]) {

        width: 500px !important;

        max-width: 500px !important;

        min-width: 500px !important;

    }

    div[data-testid="stPlotlyChart"]:has(.js-plotly-plot[data-unformatted\*="daily\_deaths\_chart"]) {

        width: 500px !important;

        max-width: 500px !important;

        min-width: 500px !important;

    }

    .daily-deaths-chart-container {

        width: 500px !important;

        max-width: 500px !important;

        min-width: 500px !important;

        margin: 10px 0;

    }

    .daily-deaths-chart-container .stElementContainer,

    .daily-deaths-chart-container .element-container,

    .daily-deaths-chart-container div[data-testid="stPlotlyChart"] {

        width: 500px !important;

        max-width: 500px !important;

        min-width: 500px !important;

    }

    h3 {

        font-size: 1.2em !important;

        margin-bottom: 10px !important;

        margin-top: 20px !important;

        font-weight: 600;

        color: #161616;

        border-bottom: 2px solid #0F62FE;

        padding-bottom: 5px;

    }

    .section-title {

        font-size: 0.9em !important;

        margin-bottom: 3px !important;

        margin-top: 2px !important;

        font-weight: 600;

        color: #161616;

        border-bottom: none !important;

        padding-bottom: 0 !important;

        text-align: left !important;

    }

    .stTabs [data-baseweb="tab-panel"] h3 {

        margin-top: 10px !important;

    }

    h6 {

        font-size: 0.9em !important;

        color: #525252;

        text-align: center;

        font-weight: 600;

        margin-bottom: 5px;

        margin-top: 5px;

    }

    .stButton>button {

        border-radius: 4px;

        padding: 8px 15px;

        border: 1px solid #0F62FE;

        background-color: #FFFFFF;

        color: #0F62FE;

        transition: all 0.3s ease;

    }

    .stButton>button:hover {

        background-color: #0F62FE;

        color: #FFFFFF;

        border: 1px solid #0043CE;

    }

    .stButton>button:active {

        background-color: #0031A1 !important;

        color: #FFFFFF !important;

    }

    .casualty-kpi-card:hover {

        transform: scale(1.03);

        box-shadow: 0 5px 15px rgba(15, 98, 254, 0.3);

        cursor: pointer;

    }

    .casualty-kpi-card.death-total {

        border-left-color: #DA1E28;

    }

    .casualty-kpi-card.death-total .value {

        color: #DA1E28;

    }

    .gauge-overall-container {

        background-color: #FFFFFF;

        padding: 8px;

        border-radius: 8px;

        box-shadow: 0 2px 5px rgba(0,0,0,0.1);

        margin-bottom: 15px;

        display: flex;

        align-items: stretch;

    }

    .gauge-container {

        background-color: #FFFFFF;

        padding: 3px;

        border-radius: 8px;

        box\_shadow: 0 2px 5px rgba(0,0,0,0.1);

        margin-bottom: 10px;

        animation: fadeIn 0.8s ease-out forwards;

        transition: transform 0.2s ease-in-out, box-shadow 0.2s ease-in-out, opacity 0.8s ease-out;

        display: flex;

        flex-direction: column;

        align-items: center;

        justify-content: center;

        text-align: center;

        width: 100%;

        box-sizing: border-box;

    }

    .gauge-container:hover {

        transform: translateY(-5px) scale(1.02);

        box-shadow: 0 6px 12px rgba(15, 98, 254, 0.25);

        cursor: default;

    }

    .gauge-container .stPlotlyChart {

        padding: 0 !important;

        margin: 0 !important;

        width: 100%;

        height: 100%;

        display: flex;

        justify-content: center;

        align-items: center;

    }

    .gauge-value {

        font-size: 1.4em;

        font-weight: 700;

        color: #161616;

        margin-top: 5px;

        text-align: center;

        width: 100%;

    }

    .st-emotion-cache-1l269bu {

        background-color: #FFFFFF;

        border-right: 1px solid #E0E0E0;

        box-shadow: 2px 0 5px rgba(0,0,0,0.05);

        padding-top: 1.5rem;

        padding-left: 0.8rem;

        padding-right: 0.8rem;

    }

    .st-emotion-cache-1l269bu .stSelectbox, .st-emotion-cache-1l269bu .stDateInput {

        margin-bottom: 10px;

    }

    .sidebar-title {

        font-size: 1.2em;

        font-weight: 600;

        color: #0F62FE;

        margin-bottom: 15px;

        text-align: center;

    }

    button[data-baseweb="tab"] {

        font-size: 0.9em !important;

        padding: 8px 15px !important;

        font-family: "IBM Plex Sans", sans-serif !important;

        font-weight: 500 !important;

    }

    [data-baseweb="tab-list"] {

        background-color: #F4F4F4 !important;

        padding-bottom: 0px !important;

        margin-bottom: 10px;

    }

    .gauge-overall-container > div[data-testid="stHorizontalBlock"] {

        margin: 0 !important;

        padding: 0 !important;

        gap: 0px !important;

    }

    .gauge-overall-container div[data-testid="stColumn"] {

        padding-left: 0rem !important;

        padding-right: 0rem !important;

        padding-top: 0rem !important;

        padding-bottom: 0rem !important;

        margin: 0 !important;

        border-right: 1px solid #E0E0E0;

        box-sizing: border-box;

    }

    .gauge-overall-container div[data-testid="stColumn"]:last-child {

        border-right: none;

    }

    .gauge-overall-container div[data-testid="stColumn"] > div {

        padding: 0 ;

        margin: 0 ;

    }

    .incident-summary-wrapper-container div[data-testid="stHorizontalBlock"] {

        max-width: 100%;

        padding: 0px ;

        margin: 0px ;

        gap: 2px ;

        width: 100% ;

    }

    .incident-summary-wrapper-container div[data-testid="stColumn"] {

        padding: 0px 1px ;

        margin: 0px ;

        display: flex;

        flex-direction: column;

        align-items: center;

        justify-content: flex-start;

        flex-grow: 1;

    }

    .incident-death-card {

        background-color: #6777EF;

        color: white;

        padding: 1px 6px;

        border-radius: 5px;

        text-align: center;

        margin: 2px 1px;

        font-weight: 500;

        height: 22px;

        display: flex;

        flex-direction: row;

        justify-content: space-between;

        align-items: center;

        border: 1px solid transparent;

        box-shadow: 0 1px 4px rgba(103,119,239,0.10);

        width: 100%;

        max-width: 140px;

        white-space: nowrap;

        overflow: hidden;

        transition: background-color 0.3s ease, border-color 0.3s ease, box-shadow 0.15s, transform 0.12s;

        text-overflow: ellipsis;

        box-sizing: border-box;

        font-size: 0.85em;

    }

    .incident-death-card:hover {

        background-color: #3B5BDB;

        border: 1px solid #000000;

        box-shadow: 0 4px 12px rgba(0,0,0,0.25);

        transform: translateY(-2px) scale(1.03);

    }

    .incident-death-card.total-deaths-card {

        background-color: #FC403B;

        border: 1px solid transparent;

        transition: background-color 0.3s ease, border-color 0.3s ease, box-shadow 0.15s, transform 0.12s;

        box-shadow: 0 1px 4px rgba(252,84,75,0.10);

    }

    .incident-death-card.total-deaths-card:hover {

        background-color: #DA1E28;

        border: 1px solid #000000;

        box-shadow: 0 4px 12px rgba(0,0,0,0.25);

        transform: translateY(-2px) scale(1.03);

    }

    .incident-death-card .incident-label {

        font-size: 0.8em;

        font-weight: 700;

        font-family: 'Segoe UI', 'Roboto', 'Arial', sans-serif;

        letter-spacing: 0.3px;

        text-rendering: optimizeLegibility;

        -webkit-font-smoothing: antialiased;

        -moz-osx-font-smoothing: grayscale;

        white-space: nowrap;

        overflow: hidden;

        text-overflow: ellipsis;

        flex-shrink: 1;

        margin-right: 3px;

        line-height: 1;

    }

    .incident-death-card .incident-value {

        font-size: 0.9em;

        font-weight: 600;

        font-family: 'Segoe UI', 'Roboto', 'Arial', sans-serif;

        letter-spacing: 0.4px;

        text-rendering: optimizeLegibility;

        -webkit-font-smoothing: antialiased;

        -moz-osx-font-smoothing: grayscale;

        flex-shrink: 0;

        white-space: nowrap;

        line-height: 1;

    }

    .st-emotion-cache-1f8u01d {

        background-color: #F8FAFF;

        border-radius: 12px;

        border: 1.5px solid #E0E6F0;

        box-shadow: 0 2px 8px rgba(0,0,0,0.07);

        padding: 15px 8px 10px 8px;

        margin-bottom: 18px;

        display: flex;

        flex-wrap: wrap;

        justify-content: flex-start;

        align-items: flex-start;

        align-content: flex-start;

        gap: 0px;

        height: auto;

        max-height: 320px;

        overflow-y: auto;

        width: 100%;

        max-width: 100%;

        margin-left: 0;

        margin-right: 0;

    }

    .incident-summary-wrapper-container > div {

        padding: 0 !important;

    }

    .incident-summary-wrapper-container .st-emotion-cache-1f8u01d {

        width: 100%;

    }

    .st-emotion-cache-1d8vwwt.e1lln2w84 {

        background-color: #F8FAFF;

        border-radius: 12px;

        border: 1.5px solid #E0E6F0;

        box-shadow: 0 2px 8px rgba(0,0,0,0.07);

        padding: 15px 8px 10px 8px;

        margin-bottom: 18px;

        display: flex;

        flex-wrap: wrap;

        justify-content: flex-start;

        align-items: flex-start;

        align-content: flex-start;

        gap: 0px;

        height: auto;

        max-height: 320px;

        overflow-y: auto;

        width: 320px !important; /\* Reduced width for more compact container \*/

        max-width: 320px !important; /\* Set maximum width \*/

        margin-left: 0;

        margin-right: 0;

    }

    .incident-summary-wrapper-container .stElementContainer.element-container.st-emotion-cache-17lr0tt.e1lln2w81 {

        width: 100% !important; /\* Use full column width for professional alignment \*/

        max-width: 100% !important;

        margin-left: 0 !important;

        margin-right: 0 !important;

    }

    .incident-summary-wrapper-container div.st-emotion-cache-17lr0tt.e1lln2w81 {

        width: 100% !important;

        max-width: 100% !important;

        margin-left: 0 !important;

        margin-right: 0 !important;

    }

    .stPlotlyChart {

        background-color: transparent !important;

        box-shadow: none !important;

        border-radius: 0 !important;

        padding: 0 !important;

        margin-top: 0px !important;

        margin-bottom: 0px !important;

    }

    div[data-testid="column"] {

        vertical-align: top !important;

        align-items: flex-start !important;

    }

    div[data-testid="column"] > div {

        vertical-align: top !important;

        align-items: flex-start !important;

        display: flex !important;

        flex-direction: column !important;

        justify-content: flex-start !important;

    }

    .js-plotly-plot {

        margin-top: 0 !important;

        padding-top: 0 !important;

    }

    div[data-testid="column"] h6,

    div[data-testid="column"] h3 {

        margin-top: 0 !important;

        padding-top: 0 !important;

        margin-bottom: 10px !important;

    }

    div[data-testid="column"] .stMarkdown,

    div[data-testid="column"] .stContainer {

        margin-top: 0 !important;

        padding-top: 0 !important;

    }

    div[data-testid="stPlotlyChart"] {

        transition: none !important;

    }

    div[data-testid="stPlotlyChart"] > div {

        transition: none !important;

    }

    .js-plotly-plot .plotly {

        transition: none !important;

    }

    .footer-card { background-color: #343a40; color: #dee2e6; padding: 1rem; border-radius: 8px; margin-top: 1.5rem; text-align: center; box-shadow: 0 -1px 3px rgba(0,0,0,0.05); border-top: 1px solid var(--card-border-color); font-size: 0.8rem; box-sizing: border-box; }

    .footer-title { color: #ffc107 !important; font-size: 1rem !important; font-weight: 500 !important; margin-bottom: 0.5rem !important; }

    .footer-card p { margin-bottom: 0.25rem; font-size: 0.75rem;}

</style>

<script>

document.addEventListener('DOMContentLoaded', function() {

    let resizeTimeout;

    // Function to trigger Plotly resize

    function resizePlotlyCharts() {

        const plotlyCharts = document.querySelectorAll('.js-plotly-plot');

        plotlyCharts.forEach(chart => {

            if (window.Plotly && chart.\_fullLayout) {

                window.Plotly.Plots.resize(chart);

            }

        });

    }

    // Monitor for sidebar state changes

    const observer = new MutationObserver(function(mutations) {

        mutations.forEach(function(mutation) {

            if (mutation.type === 'attributes' &&

                (mutation.attributeName === 'class' || mutation.attributeName === 'style')) {

                clearTimeout(resizeTimeout);

                resizeTimeout = setTimeout(resizePlotlyCharts, 100);

            }

        });

    });

    const sidebar = document.querySelector('[data-testid="stSidebar"]');

    if (sidebar) {

        observer.observe(sidebar, { attributes: true, subtree: true });

    }

    window.addEventListener('resize', function() {

        clearTimeout(resizeTimeout);

        resizeTimeout = setTimeout(resizePlotlyCharts, 100);

    });

});

</script>

    """, unsafe\_allow\_html=True)

    PRIMARY\_COLOR = "#0F62FE"

    SECONDARY\_COLOR = "#525252"

    DEATH\_COLOR = "#DA1E28"

    INJURED\_COLOR = "#FF8C00"

    INCIDENT\_COLOR = PRIMARY\_COLOR

    TEXT\_COLOR = "#161616"

    GRID\_COLOR = "#E0E0E0"

    AXIS\_COLOR = "#C6C6C6"

    PLOTLY\_TEMPLATE = "plotly\_white"

    CHART\_FONT = "IBM Plex Sans, sans-serif"

    # Image to base64 conversion function

    @st.cache\_data

    def get\_image\_as\_base64(path):

        try:

            with open(path, "rb") as image\_file:

                encoded\_string = base64.b64encode(image\_file.read()).decode()

            image\_type = path.split('.')[-1].lower()

            if image\_type in ["jpg", "jpeg"]: return f"data:image/jpeg;base64,{encoded\_string}"

            elif image\_type == "png": return f"data:image/png;base64,{encoded\_string}"

            else: return f"data:image/png;base64,{encoded\_string}"

        except FileNotFoundError:

            st.warning(f"Image file not found at path: '{path}'. Please ensure it is in the correct directory.")

            return "data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7"

    # Database Connection

    @st.cache\_resource

    def init\_db\_connection():

        """Establishes a SQLAlchemy engine connection to the SQL Server database using Dashboard2.toml config."""

        try:

            from sqlalchemy import create\_engine

            import os

            config\_path = os.path.join(".streamlit", "Dashboard2.toml")

            if os.path.exists(config\_path):

                # Creating SQLAlchemy connection string

                connection\_url = "mssql+pyodbc://KAKA/eoc?driver=ODBC+Driver+17+for+SQL+Server&trusted\_connection=yes&TrustServerCertificate=yes"

            else:

                # Fallback connection string

                connection\_url = "mssql+pyodbc://KAKA/eoc?driver=ODBC+Driver+17+for+SQL+Server&trusted\_connection=yes&TrustServerCertificate=yes"

            # Add connection pooling for better performance

            engine = create\_engine(

                connection\_url,

                pool\_size=5,

                max\_overflow=10,

                pool\_pre\_ping=True,

                pool\_recycle=3600

            )

            return engine

        except Exception as e:

            st.error(f"Database connection failed. Check `Dashboard2.toml` and ensure DB is running. Error: {e}")

            return None

    # Data Loading - Extended cache for better performance

    @st.cache\_data(ttl=1800)  # 30 minutes instead of 10

    def load\_data\_from\_db():

        try:

            engine = init\_db\_connection()

            if not engine:

                return pd.DataFrame()

            sql\_query = """

            SELECT

                CAST(HR.IncidentDate AS DATE) AS date,

                TRIM(MD.DistrictName) AS district,

                TRIM(MB.BlockName) AS block,

                H.Name AS incident\_type,

                COALESCE(SUM(CASE WHEN HLR.HLCode = 2 THEN 1 ELSE 0 END), 0) AS deaths, -- Corrected: HLCode = 2 for Deaths

                COALESCE(SUM(CASE WHEN HLR.HLCode = 1 THEN 1 ELSE 0 END), 0) AS injured, -- Corrected: HLCode = 1 for Injured

                CASE

                    WHEN HR.IsFinal = 1 THEN 'Final'

                    WHEN HR.IsFinal = 2 THEN 'Verified'

                    ELSE 'Unknown'

                END AS entry\_type

            FROM

                dbo.HazardReport AS HR

            LEFT JOIN

                dbo.Hazards AS H ON HR.HazardCode = H.ID

            LEFT JOIN

                dbo.mst\_Districts AS MD ON HR.DistrictCode = MD.DistrictCode

            LEFT JOIN

                dbo.mst\_Blocks AS MB ON HR.BlockCode = MB.BlockCode AND HR.DistrictCode = MB.DistrictCode

            LEFT JOIN

                dbo.HumanLossReport AS HLR ON HR.ID = HLR.HzdReptID

            WHERE

                HR.IncidentDate >= DATEADD(YEAR, -5, GETDATE())  -- Last 5 years only for faster loading

            GROUP BY

                CAST(HR.IncidentDate AS DATE),

                TRIM(MD.DistrictName),

                TRIM(MB.BlockName),

                H.Name,

                HR.IsFinal

            ORDER BY

                CAST(HR.IncidentDate AS DATE);

            """

            df = pd.read\_sql(sql\_query, engine)

            if df.empty:

                return pd.DataFrame()

            # Highly optimized data processing - vectorized operations

            df['date'] = pd.to\_datetime(df['date'], format='%Y-%m-%d', errors='coerce')

            # Vectorized string processing - much faster

            string\_cols = ['district', 'block', 'incident\_type', 'entry\_type']

            df[string\_cols] = df[string\_cols].astype(str).apply(lambda x: x.str.strip().str.title())

            # Vectorized numeric processing

            numeric\_cols = ['deaths', 'injured']

            df[numeric\_cols] = df[numeric\_cols].apply(pd.to\_numeric, errors='coerce').fillna(0).astype('int16')

            # Remove any duplicate rows to reduce memory

            df = df.drop\_duplicates().reset\_index(drop=True)

            return df

        except Exception as e:

            st.error(f"An error occurred while loading data: {e}. Please ensure:")

            st.error("- Your `.streamlit/Dashboard2.toml` file includes the correct server and database configuration.")

            st.error("- The database server is accessible and the ODBC Driver 17 for SQL Server is installed.")

            return pd.DataFrame()

    # Cache filtered data to avoid repeated processing

    @st.cache\_data

    def get\_filtered\_data(df, start\_date, end\_date, district, entry\_type, incident\_type):

        """Cache filtered data based on filter selections"""

        df\_filtered = df[(df['date'] >= start\_date) & (df['date'] <= end\_date)].copy()

        if district != 'All':

            df\_filtered = df\_filtered[df\_filtered['district'] == district]

        if entry\_type != 'All':

            df\_filtered = df\_filtered[df\_filtered['entry\_type'] == entry\_type]

        if incident\_type != 'All':

            df\_filtered = df\_filtered[df\_filtered['incident\_type'] == incident\_type]

        # Apply incident type replacement

        if 'incident\_type' in df\_filtered.columns:

            df\_filtered['incident\_type'] = df\_filtered['incident\_type'].replace('Strong Wind (Andhi Toofan)', 'Strong Wind')

        return df\_filtered

    df\_main = load\_data\_from\_db()

    if df\_main.empty: st.stop()

    # Main Page Header

    eoc\_logo\_header\_base64 = get\_image\_as\_base64("eoc\_logo.png")

    header\_logo\_html = f'<img src="{eoc\_logo\_header\_base64}" alt="EOC" class="header-logo-img">' if eoc\_logo\_header\_base64 else ""

    st.markdown(f"""

        <div class="dashboard-header">

            {header\_logo\_html}

            <div class="header-title-block">

                <h1>Incident Dashboard</h1>

                <p class="tagline">Emergency Operations Center - Government of Bihar</p>

            </div>

        </div>

    """, unsafe\_allow\_html=True)

    min\_date\_data = df\_main['date'].min().date()

    max\_date\_data = df\_main['date'].max().date()

    default\_start\_date\_filter = datetime(2021, 1, 1).date()

    default\_end\_date\_filter = datetime(2025, 6, 30).date()

    if default\_start\_date\_filter < min\_date\_data:

        default\_start\_date\_filter = min\_date\_data

    if default\_end\_date\_filter > max\_date\_data:

        default\_end\_date\_filter = max\_date\_data

    col1, col2, col3, col4 = st.columns([1.2, 1, 1, 1])

    with col1:

        date\_range\_tuple\_filter = st.date\_input(

            "Select Date Range",

            value=(default\_start\_date\_filter, default\_end\_date\_filter),

            min\_value=min\_date\_data,

            max\_value=max\_date\_data,

            key="date\_range\_filter\_sidebar"

        )

    with col2:

        selected\_district = st.selectbox(

            "Select District",

            ['All'] + sorted(list(df\_main['district'].unique())),

            key="district\_filter\_sidebar"

        )

    with col3:

        selected\_entry\_type = st.selectbox(

            "Select Entry Type",

            ['All'] + sorted(list(df\_main['entry\_type'].unique())),

            key="entry\_type\_filter\_sidebar"

        )

    with col4:

        selected\_incident\_type = st.selectbox(

            "Select Incident Type",

            ['All'] + sorted(list(df\_main['incident\_type'].unique())),

            key="incident\_type\_filter\_sidebar"

        )

    if len(date\_range\_tuple\_filter) == 2:

        start\_date, end\_date = pd.to\_datetime(date\_range\_tuple\_filter[0]), pd.to\_datetime(date\_range\_tuple\_filter[1])

    else:

        start\_date, end\_date = pd.to\_datetime(default\_start\_date\_filter), pd.to\_datetime(default\_end\_date\_filter)

        st.warning("Please select a valid date range (start and end date).")

    if start\_date > end\_date:

        st.error("Error: Start date cannot be after end date. Please adjust the date range.")

        st.stop()

    # Use cached filtering for better performance

    df\_filtered = get\_filtered\_data(df\_main, start\_date, end\_date, selected\_district, selected\_entry\_type, selected\_incident\_type)

    total\_incidents = len(df\_filtered)

    total\_deaths = df\_filtered['deaths'].sum() if 'deaths' in df\_filtered else 0

    total\_injured = df\_filtered['injured'].sum() if 'injured' in df\_filtered else 0

    @st.cache\_data

    def create\_plotly\_gauge\_figure(value, title\_text, color, max\_value):

        fig = go.Figure()

        fig.add\_trace(go.Indicator(

            mode="gauge+number",

            value=value,

            number={

                'font': {'size': 24, 'family': CHART\_FONT, 'color': color},

                'valueformat': ',d',

            },

            title={

                'text': f'<span style="font-size:18px; font-weight:500; color:{color}; text-decoration:underline; text-decoration-color:{color}; text-decoration-thickness:1px;">{title\_text}</span>',  # Bigger title with medium weight

                'font': {'size': 18, 'family': CHART\_FONT, 'color': color}

            },

            gauge={

                'axis': {'range': [None, max\_value], 'tickwidth': 1, 'tickcolor': "darkgray"},

                'bar': {'color': color, 'thickness': 0.75},

                'bgcolor': "white",

                'borderwidth': 1,

                'bordercolor': "#E0E0E0",

                'steps': [

                    {'range': [0, max\_value \* 0.5], 'color': "#F0F0F0"},

                    {'range': [max\_value \* 0.5, max\_value], 'color': "#E0E0E0"}

                ]

            }

        ))

        fig.update\_layout(

            height=140,

            margin=dict(l=5, r=5, t=50, b=5),

            paper\_bgcolor='rgba(0,0,0,0)',

            plot\_bgcolor='rgba(0,0,0,0)',

            font\_family=CHART\_FONT,

            font\_color=TEXT\_COLOR,

        )

        return fig

    INCIDENTS\_GAUGE\_MAX = max(10, total\_incidents + int(total\_incidents\*0.25)) if total\_incidents > 0 else 10

    DEATHS\_GAUGE\_MAX = max(10, total\_deaths + int(total\_deaths\*0.5) + 5) if total\_deaths > 0 else 10

    INJURED\_GAUGE\_MAX = max(10, total\_injured + int(total\_injured\*0.5) + 5) if total\_injured > 0 else 10

    gauge\_key\_base = f"{total\_incidents}\_{total\_deaths}\_{total\_injured}\_{start\_date.strftime('%Y%m%d')}\_{end\_date.strftime('%Y%m%d')}\_{selected\_district}\_{selected\_entry\_type}\_{selected\_incident\_type}"

    with st.markdown('<div class="gauge-overall-container">', unsafe\_allow\_html=True):

        kpi\_col1, kpi\_col2, kpi\_col3 = st.columns(3)

        with kpi\_col1:

            fig\_incidents = create\_plotly\_gauge\_figure(total\_incidents, "Incidents", INCIDENT\_COLOR, INCIDENTS\_GAUGE\_MAX)

            st.plotly\_chart(fig\_incidents, use\_container\_width=True, config={'displayModeBar': False}, key=f"incidents\_chart\_{gauge\_key\_base}")

        with kpi\_col2:

            fig\_deaths = create\_plotly\_gauge\_figure(total\_deaths, "Deaths", DEATH\_COLOR, DEATHS\_GAUGE\_MAX)

            st.plotly\_chart(fig\_deaths, use\_container\_width=True, config={'displayModeBar': False}, key=f"deaths\_chart\_{gauge\_key\_base}")

        with kpi\_col3:

            fig\_injured = create\_plotly\_gauge\_figure(total\_injured, "Injured", INJURED\_COLOR, INJURED\_GAUGE\_MAX)

            st.plotly\_chart(fig\_injured, use\_container\_width=True, config={'displayModeBar': False}, key=f"injured\_chart\_{gauge\_key\_base}")

    st.markdown('<hr style="border: none; height: 1px; background: linear-gradient(90deg, #0F62FE 0%, #4589FF 100%); margin: 6px 0 8px 0; opacity: 0.7;">', unsafe\_allow\_html=True)

    def style\_plotly\_chart(fig, chart\_height=280, is\_pie\_or\_donut=False, legend\_orientation="h"):

        fig.update\_layout(

            template=PLOTLY\_TEMPLATE,

            plot\_bgcolor='rgba(0,0,0,0)',

            paper\_bgcolor='rgba(0,0,0,0)',

            xaxis=dict(

                showgrid=False,

                linecolor=AXIS\_COLOR,

                tickfont=dict(color=TEXT\_COLOR, size=10),

                showline=True,

                zeroline=False

            ),

            yaxis=dict(

                showgrid=True,

                gridcolor=GRID\_COLOR,

                linecolor=AXIS\_COLOR,

                tickfont=dict(color=TEXT\_COLOR, size=10),

                showline=False,

                zeroline=False,

                rangemode='tozero'

            ),

            hoverlabel=dict(bgcolor="#FFFFFF", font\_size=11, font\_family=CHART\_FONT, bordercolor=AXIS\_COLOR, font\_color=TEXT\_COLOR),

            legend=dict(orientation=legend\_orientation, yanchor="bottom", y=1.02, xanchor="right", x=1, font=dict(family=CHART\_FONT, color=TEXT\_COLOR, size=10), bgcolor='rgba(0,0,0,0)'),

            margin=dict(l=20, r=10, t=30, b=30),

            title\_text='',

            title\_font\_family=CHART\_FONT,

            title\_font\_color=TEXT\_COLOR,

            font\_color=TEXT\_COLOR,

            font\_family=CHART\_FONT,

            height=chart\_height

        )

        if not is\_pie\_or\_donut:

            fig.update\_layout(hovermode='x unified')

            fig.update\_xaxes(title\_text=None)

            fig.update\_yaxes(title\_text=None)

        else:

            fig.update\_layout(hovermode='closest')

            fig.update\_layout(

                xaxis=dict(visible=False, showgrid=False),

                yaxis=dict(visible=False, showgrid=False),

                margin=dict(l=5, r=5, t=25, b=5)

            )

        if fig.layout.title.text:

            fig.update\_layout(title\_x=0.5)

        return fig

    graph\_col1, graph\_col2, graph\_col3 = st.columns([1, 1, 1], gap="small")

    with graph\_col1:

        st.markdown('<h3 class="section-title">Casualties</h3>', unsafe\_allow\_html=True)

        if not df\_filtered.empty and 'deaths' in df\_filtered.columns and 'incident\_type' in df\_filtered.columns:

            incident\_deaths\_summary = df\_filtered[df\_filtered['deaths'] > 0].groupby('incident\_type')['deaths'].sum().sort\_values(ascending=False).reset\_index()

            st.markdown('<div class="incident-summary-wrapper-container">', unsafe\_allow\_html=True)

            incident\_cards\_container = st.container(border=True)

            with incident\_cards\_container:

                if not incident\_deaths\_summary.empty:

                    all\_death\_summary\_items = incident\_deaths\_summary.to\_dict('records')

                    all\_death\_summary\_items.append({"incident\_type": "Total Deaths", "deaths": total\_deaths, "is\_total": True})

                    cols = st.columns([1, 1])

                    left\_column\_items = [item for i, item in enumerate(all\_death\_summary\_items) if i % 2 == 0]

                    right\_column\_items = [item for i, item in enumerate(all\_death\_summary\_items) if i % 2 != 0]

                    with cols[0]:

                        for item in left\_column\_items:

                            card\_class = "incident-death-card"

                            if item.get("is\_total"):

                                card\_class += " total-deaths-card"

                            st.markdown(f"""

                                <div class="{card\_class}">

                                    <span class="incident-label">{item["incident\_type"]} :</span>

                                    <span class="incident-value">{item["deaths"]:,}</span>

                                </div>

                            """, unsafe\_allow\_html=True)

                    with cols[1]:

                        for item in right\_column\_items:

                            card\_class = "incident-death-card"

                            if item.get("is\_total"):

                                card\_class += " total-deaths-card"

                            st.markdown(f"""

                                <div class="{card\_class}">

                                    <span class="incident-label">{item["incident\_type"]} :</span>

                                    <span class="incident-value">{item["deaths"]:,}</span>

                                </div>

                            """, unsafe\_allow\_html=True)

                else:

                    incident\_cards\_container.markdown('<p style="text-align: center; color: #525252; padding-top: 50px;">No deaths by incident type for selected filters.</p>', unsafe\_allow\_html=True)

        else:

            st.info("Required columns for 'Casualties' (deaths, incident\_type) are not available or no data in filtered set.")

    try:

        if not df\_filtered.empty and 'deaths' in df\_filtered.columns and 'date' in df\_filtered.columns:

            selected\_month = end\_date.month

            selected\_year = end\_date.year

            month\_data = df\_filtered[

                (df\_filtered['date'].dt.month == selected\_month) &

                (df\_filtered['date'].dt.year == selected\_year)

            ].copy()

            if not month\_data.empty and month\_data['deaths'].sum() > 0:

                daily\_deaths = month\_data.groupby('date')['deaths'].sum().reset\_index()

                daily\_deaths = daily\_deaths.sort\_values('date')

            else:

                import pandas as pd

                start\_of\_month = datetime(selected\_year, selected\_month, 1)

                if selected\_month == 12:

                    end\_of\_month = datetime(selected\_year + 1, 1, 1) - timedelta(days=1)

                else:

                    end\_of\_month = datetime(selected\_year, selected\_month + 1, 1) - timedelta(days=1)

                # Creating date range for the month

                date\_range = pd.date\_range(start=start\_of\_month, end=end\_of\_month, freq='D')

                daily\_deaths = pd.DataFrame({

                    'date': date\_range,

                    'deaths': [0] \* len(date\_range)

                })

            # Creating line chart with smooth curves

            fig\_daily\_deaths = px.line(

                daily\_deaths,

                x='date',

                y='deaths',

                title="Current Month",

                labels={'date': 'Date', 'deaths': 'Number of Deaths'},

                color\_discrete\_sequence=['#3C6FF7'],

                markers=True

            )

            fig\_daily\_deaths.update\_traces(

                    mode='lines+markers',

                    line=dict(

                        width=3,

                        color='#3C6FF7',

                        shape='spline',

                        smoothing=0.3

                    ),

                    marker=dict(

                        size=8,

                        color='white',

                        symbol='circle',

                        line=dict(color='#3C6FF7', width=2)

                    ),

                    hovertemplate='%{x}<br>Deaths: %{y}<extra></extra>'

                )

            fig\_daily\_deaths.update\_layout(

                    width=500,

                    height=288,

                    autosize=False,

                    template=PLOTLY\_TEMPLATE,

                    plot\_bgcolor='rgba(0,0,0,0)',

                    paper\_bgcolor='rgba(0,0,0,0)',

                    title=dict(

                        text='Current Month',

                        font=dict(size=12, family=CHART\_FONT, color=TEXT\_COLOR, weight=600),

                        x=0.5,

                        xanchor='center'

                    ),

                    xaxis=dict(

                        title=dict(text='Date', font=dict(size=12, family=CHART\_FONT)),

                        showgrid=True,

                        gridcolor=GRID\_COLOR,

                        linecolor=AXIS\_COLOR,

                        tickfont=dict(color=TEXT\_COLOR, size=10),

                        showline=True

                    ),

                    yaxis=dict(

                        title=dict(text='Number of Deaths', font=dict(size=12, family=CHART\_FONT)),

                        showgrid=True,

                        gridcolor=GRID\_COLOR,

                        linecolor=AXIS\_COLOR,

                        tickfont=dict(color=TEXT\_COLOR, size=10),

                        showline=False,

                        rangemode='tozero'

                    ),

                    margin=dict(l=40, r=15, t=30, b=30),

                    font\_family=CHART\_FONT,

                    font\_color=TEXT\_COLOR,

                    hovermode='x unified'

                )

            chart\_col1, chart\_col2 = st.columns([1, 1], gap="small")

            with chart\_col1:

                    st.markdown('<div class="daily-deaths-chart-container">', unsafe\_allow\_html=True)

                    st.plotly\_chart(fig\_daily\_deaths, use\_container\_width=True, config={'displayModeBar': False}, key=f"daily\_deaths\_chart\_{selected\_month}\_{selected\_year}")

                    st.markdown('</div>', unsafe\_allow\_html=True)

            with chart\_col2:

                    try:

                        if not df\_filtered.empty and 'deaths' in df\_filtered.columns and 'date' in df\_filtered.columns and 'incident\_type' in df\_filtered.columns:

                            selected\_year = end\_date.year

                            if selected\_incident\_type and selected\_incident\_type != 'All':

                                selected\_incident = selected\_incident\_type

                                yearly\_incident\_data = df\_main[

                                    (df\_main['date'].dt.year == selected\_year) &

                                    (df\_main['incident\_type'] == selected\_incident)

                                ].copy()

                            else:

                                yearly\_incident\_data = df\_main[

                                    df\_main['date'].dt.year == selected\_year

                                ].copy()

                                selected\_incident = "All Incidents"

                            # Creating complete month range for the year (Jan to Dec)

                            import calendar

                            all\_months = []

                            for month\_num in range(1, 13):

                                month\_name = calendar.month\_abbr[month\_num]

                                all\_months.append({'month': month\_num, 'month\_name': month\_name, 'deaths': 0})

                            # Creating DataFrame with all 12 months

                            monthly\_deaths = pd.DataFrame(all\_months)

                            if not yearly\_incident\_data.empty:

                                yearly\_incident\_data['month'] = yearly\_incident\_data['date'].dt.month

                                actual\_monthly\_deaths = yearly\_incident\_data.groupby('month')['deaths'].sum().reset\_index()

                                for \_, row in actual\_monthly\_deaths.iterrows():

                                    monthly\_deaths.loc[monthly\_deaths['month'] == row['month'], 'deaths'] = row['deaths']

                                max\_deaths = monthly\_deaths['deaths'].max()

                                min\_deaths = monthly\_deaths['deaths'].min()

                                def get\_gradient\_color(value, min\_val, max\_val):

                                    if max\_val == min\_val:

                                        return '#4CAF50'

                                    normalized = (value - min\_val) / (max\_val - min\_val)

                                    light\_r, light\_g, light\_b = 129, 199, 132

                                    dark\_r, dark\_g, dark\_b = 46, 125, 50

                                    r = int(light\_r + (dark\_r - light\_r) \* normalized)

                                    g = int(light\_g + (dark\_g - light\_g) \* normalized)

                                    b = int(light\_b + (dark\_b - light\_b) \* normalized)

                                    return f'rgb({r},{g},{b})'

                                monthly\_deaths['color'] = monthly\_deaths['deaths'].apply(

                                    lambda x: get\_gradient\_color(x, min\_deaths, max\_deaths)

                                )

                                # Creating column chart

                                fig\_monthly\_deaths = px.bar(

                                    monthly\_deaths,

                                    x='month\_name',

                                    y='deaths',

                                    title=f"Monthly Casualties - {selected\_incident} ({selected\_year})",

                                    labels={'month\_name': 'Month', 'deaths': 'Number of Casualties'},

                                    color='deaths',

                                    color\_continuous\_scale='Greens',

                                    text='deaths'

                                )

                                fig\_monthly\_deaths.update\_traces(

                                    texttemplate='%{text}',

                                    textposition='outside',

                                    hovertemplate='Month: %{x}<br>Deaths: %{y}<extra></extra>',

                                    marker\_line\_color='rgba(0,0,0,0.3)',

                                    marker\_line\_width=1

                                )

                                fig\_monthly\_deaths.update\_layout(

                                    width=500,

                                    height=288,

                                    autosize=False,

                                    template=PLOTLY\_TEMPLATE,

                                    plot\_bgcolor='rgba(0,0,0,0)',

                                    paper\_bgcolor='rgba(0,0,0,0)',

                                    title={

                                        'text': f"Monthly Casualties - {selected\_incident} ({selected\_year})",

                                        'x': 0.5,

                                        'xanchor': 'center',

                                        'font': {'size': 12, 'color': TEXT\_COLOR, 'weight': 600}  # Match section-title styling

                                    },

                                    xaxis={

                                        'title': {'text': 'Month', 'font': {'size': 12, 'color': '#666666'}},

                                        'tickfont': {'size': 10, 'color': '#666666'},

                                        'gridcolor': 'rgba(200,200,200,0.3)',

                                        'showgrid': True

                                    },

                                    yaxis={

                                        'title': {'text': 'Number of Deaths', 'font': {'size': 12, 'color': '#666666'}},

                                        'tickfont': {'size': 10, 'color': '#666666'},

                                        'gridcolor': 'rgba(200,200,200,0.3)',

                                        'showgrid': True

                                    },

                                    showlegend=False,

                                    margin={'l': 40, 'r': 40, 't': 45, 'b': 40}

                                )

                                # Displaying the monthly deaths column chart

                                st.markdown('<div class="daily-deaths-chart-container">', unsafe\_allow\_html=True)

                                st.plotly\_chart(fig\_monthly\_deaths, use\_container\_width=True, config={'displayModeBar': False}, key=f"monthly\_deaths\_chart\_{selected\_incident}\_{selected\_year}")

                                st.markdown('</div>', unsafe\_allow\_html=True)

                        else:

                            import calendar

                            all\_months = []

                            for month\_num in range(1, 13):

                                month\_name = calendar.month\_abbr[month\_num]

                                all\_months.append({'month': month\_num, 'month\_name': month\_name, 'deaths': 0})

                            monthly\_deaths = pd.DataFrame(all\_months)

                            # Create column chart with empty data

                            fig\_monthly\_deaths = px.bar(

                                monthly\_deaths,

                                x='month\_name',

                                y='deaths',

                                title=f"Monthly Casualties - {selected\_incident} ({selected\_year})",

                                labels={'month\_name': 'Month', 'deaths': 'Number of Casualties'},

                                color='deaths',

                                color\_continuous\_scale='Greens',

                                text='deaths'

                            )

                            fig\_monthly\_deaths.update\_traces(

                                texttemplate='%{text}',

                                textposition='outside',

                                hovertemplate='Month: %{x}<br>Deaths: %{y}<extra></extra>',

                                marker\_line\_color='rgba(0,0,0,0.3)',

                                marker\_line\_width=1

                            )

                            fig\_monthly\_deaths.update\_layout(

                                width=500,

                                height=288,

                                autosize=False,

                                template=PLOTLY\_TEMPLATE,

                                plot\_bgcolor='rgba(0,0,0,0)',

                                paper\_bgcolor='rgba(0,0,0,0)',

                                title={

                                    'text': f"Monthly Casualties - {selected\_incident} ({selected\_year})",

                                    'x': 0.5,

                                    'xanchor': 'center',

                                    'font': {'size': 12, 'color': TEXT\_COLOR, 'weight': 600}

                                },

                                xaxis={

                                    'title': {'text': 'Month', 'font': {'size': 12, 'color': '#666666'}},

                                    'tickfont': {'size': 10, 'color': '#666666'},

                                    'gridcolor': 'rgba(200,200,200,0.3)',

                                    'showgrid': True

                                },

                                yaxis={

                                    'title': {'text': 'Number of Casualties', 'font': {'size': 12, 'color': '#666666'}},

                                    'tickfont': {'size': 10, 'color': '#666666'},

                                    'gridcolor': 'rgba(200,200,200,0.3)',

                                    'showgrid': True

                                },

                                margin={'l': 40, 'r': 40, 't': 45, 'b': 40}

                            )

                            # Monthly deaths column chart

                            st.markdown('<div class="daily-deaths-chart-container">', unsafe\_allow\_html=True)

                            st.plotly\_chart(fig\_monthly\_deaths, use\_container\_width=True, config={'displayModeBar': False}, key=f"monthly\_deaths\_chart\_{selected\_incident}\_{selected\_year}")

                            st.markdown('</div>', unsafe\_allow\_html=True)

                    except Exception as e:

                        st.error(f"Could not render 'Monthly Deaths by Selected Incident' chart: {e}")

    except Exception as e:

        st.error(f"Could not render 'Daily Deaths in Selected Month' chart: {e}")

    with graph\_col2:

        st.markdown('<h3 class="section-title">Casualties(%) by Incidents</h3>', unsafe\_allow\_html=True)

        try:

            if not df\_filtered.empty and 'deaths' in df\_filtered.columns and 'incident\_type' in df\_filtered.columns and df\_filtered['deaths'].sum() > 0:

                sunburst\_data\_df = df\_filtered[df\_filtered['deaths'] > 0].groupby('incident\_type')['deaths'].sum().reset\_index()

                sunburst\_data\_df = sunburst\_data\_df.sort\_values(by='deaths', ascending=False)

                if not sunburst\_data\_df.empty:

                    fig\_sunburst = px.sunburst(

                        sunburst\_data\_df,

                        path=[px.Constant("Total Deaths"), 'incident\_type'],

                        values='deaths',

                        color='incident\_type',

                        color\_discrete\_sequence=px.colors.qualitative.Pastel,

                        custom\_data=['deaths']

                    )

                    fig\_sunburst.update\_traces(

                        textinfo='label+percent root',

                        hovertemplate='<b>%{label}</b><br>Deaths: %{customdata[0]:,}<br>(%{percentRoot:.1%})<extra></extra>',

                        insidetextorientation='radial',

                        leaf\_opacity=0.9,

                        marker\_line\_width=0.5, marker\_line\_color='rgba(0,0,0,0.4)'

                    )

                    st.plotly\_chart(style\_plotly\_chart(fig\_sunburst, chart\_height=300, is\_pie\_or\_donut=True), use\_container\_width=True, key=f"sunburst\_chart\_{total\_deaths}\_{start\_date}\_{end\_date}")

                else: st.caption("No death data by incident type to display for the selected filters.")

            elif not ('deaths' in df\_filtered.columns and 'incident\_type' in df\_filtered.columns):

                st.caption("Required columns ('deaths', 'incident\_type') missing for sunburst chart.")

            else: st.caption("No deaths recorded in the selected period to display by incident type.")

        except Exception as e:

            st.error(f"Could not render 'Deaths by Incident Type' chart: {e}")

    with graph\_col3:

        st.markdown('<h3 class="section-title">Casualties in last 7 months</h3>', unsafe\_allow\_html=True)

        try:

            if not df\_filtered.empty and 'deaths' in df\_filtered.columns and 'date' in df\_filtered.columns:

                last\_7\_months\_end\_date = end\_date

                last\_7\_months\_start\_date = (last\_7\_months\_end\_date - pd.DateOffset(months=6)).replace(day=1)

                df\_7\_months = df\_filtered[

                    (df\_filtered['date'] >= last\_7\_months\_start\_date) &

                    (df\_filtered['date'] <= last\_7\_months\_end\_date)

                ].copy()

                if not df\_7\_months.empty:

                    df\_7\_months['year\_month'] = df\_7\_months['date'].dt.to\_period('M')

                    monthly\_summary\_7\_months = df\_7\_months.groupby('year\_month')['deaths'].sum().reset\_index()

                    monthly\_summary\_7\_months['month\_label'] = monthly\_summary\_7\_months['year\_month'].dt.strftime('%b %Y')

                    monthly\_summary\_7\_months = monthly\_summary\_7\_months.sort\_values('year\_month')

                    if not monthly\_summary\_7\_months.empty and monthly\_summary\_7\_months['deaths'].sum() > 0:

                        max\_deaths = monthly\_summary\_7\_months['deaths'].max()

                        min\_deaths = monthly\_summary\_7\_months['deaths'].min()

                        def get\_red\_gradient\_color(value, min\_val, max\_val):

                            if max\_val == min\_val:

                                return '#DA1E28'

                            normalized = (value - min\_val) / (max\_val - min\_val)

                            light\_r, light\_g, light\_b = 255, 182, 193

                            dark\_r, dark\_g, dark\_b = 218, 30, 40

                            r = int(light\_r + (dark\_r - light\_r) \* normalized)

                            g = int(light\_g + (dark\_g - light\_g) \* normalized)

                            b = int(light\_b + (dark\_b - light\_b) \* normalized)

                            return f'rgb({r},{g},{b})'

                        # Applying gradient colors

                        monthly\_summary\_7\_months['color'] = monthly\_summary\_7\_months['deaths'].apply(

                            lambda x: get\_red\_gradient\_color(x, min\_deaths, max\_deaths)

                        )

                        fig\_7\_months = px.bar(

                            monthly\_summary\_7\_months,

                            x='month\_label',

                            y='deaths',

                            labels={'month\_label': 'Month', 'deaths': 'Total Deaths'},

                            color='deaths',

                            color\_continuous\_scale='Reds',

                            title=f"Total Deaths from {monthly\_summary\_7\_months['month\_label'].iloc[0]} to {monthly\_summary\_7\_months['month\_label'].iloc[-1]}" if not monthly\_summary\_7\_months.empty else "Deaths in Last 7 Months",

                            text='deaths'

                        )

                        fig\_7\_months.update\_traces(

                            texttemplate='%{text}',

                            textposition='outside',

                            hovertemplate='Month: %{x}<br>Deaths: %{y}<extra></extra>',

                            marker\_line\_color='rgba(0,0,0,0.3)',

                            marker\_line\_width=1

                        )

                        fig\_7\_months = style\_plotly\_chart(fig\_7\_months, chart\_height=330)  # Adjusted height for optimal space utilization

                        st.plotly\_chart(fig\_7\_months, use\_container\_width=True, config={'displayModeBar': False}, key=f"deaths\_7\_months\_chart\_{gauge\_key\_base}")

                    else:

                        st.caption("No deaths recorded in the last 7 months for the selected filters.")

                else:

                    st.caption("No data available for the last 7 months.")

            elif not ('deaths' in df\_filtered.columns and 'date' in df\_filtered.columns):

                st.caption("Required columns ('date', 'deaths') missing for last 7 months death report.")

            else:

                st.caption("No data available in df\_filtered to display last 7 months report.")

        except Exception as e:

            st.error(f"Could not render 'Deaths in Last 7 Months' chart: {e}")

    # Add compact divider line above the treemap section - minimal spacing

    st.markdown('<hr style="border: none; height: 1px; background: linear-gradient(90deg, #0F62FE 0%, #4589FF 100%); margin: 2px 0 6px 0; opacity: 0.7;">', unsafe\_allow\_html=True)

    st.markdown('<h3 class="section-title">District and Block wise Incident Distribution</h3>', unsafe\_allow\_html=True)

    treemap\_col = st.container()

    with treemap\_col:

        if not df\_filtered.empty and 'district' in df\_filtered.columns and 'incident\_type' in df\_filtered.columns:

            df\_treemap = df\_filtered.groupby(['district', 'incident\_type']).size().reset\_index(name='incident\_count')

            df\_treemap = df\_treemap[df\_treemap['incident\_count'] > 0]

            if not df\_treemap.empty:

                try:

                    base\_pastel\_colors = [

                        '#FBB4AE', '#B3CDE3', '#CCEBC5', '#DECBE4', '#FED9A6',

                        '#FFFFCC', '#E5D8BD', '#FDDAEC', '#F2F2F2', '#B3E2CD'

                    ]

                    shiny\_pastel\_colors = []

                    for i, color in enumerate(base\_pastel\_colors):

                        if i % 3 == 0:

                            if color == '#FBB4AE':

                                shiny\_pastel\_colors.append('#FFB3BA')

                            elif color == '#DECBE4':

                                shiny\_pastel\_colors.append('#E6D3F7')

                            elif color == '#F2F2F2':

                                shiny\_pastel\_colors.append('#F8F8FF')

                            else:

                                shiny\_pastel\_colors.append(color)

                        else:

                            shiny\_pastel\_colors.append(color)

                    fig\_treemap = px.treemap(

                        df\_treemap,

                        path=[px.Constant("All Incidents"), 'district', 'incident\_type'],

                        values='incident\_count',

                        color='district',

                        hover\_name='incident\_type',

                        custom\_data=['district'],

                        color\_discrete\_sequence=shiny\_pastel\_colors

                    )

                    fig\_treemap.update\_traces(

                        textinfo="label+value+percent parent",

                        marker=dict(

                            cornerradius=5,

                            line=dict(width=0.5, color='black')

                        ),

                        hovertemplate='<b>%{label}</b><br>District: %{customdata[0]}<br>Incidents: %{value}<br>Percentage of Parent: %{percentParent:.1%}<extra></extra>',

                        tiling=dict(

                            packing="squarify",

                            squarifyratio=1.2,

                            pad=2

                        )

                    )

                    fig\_treemap.data[0].textfont.size = 9

                    fig\_treemap.data[0].textfont.family = CHART\_FONT

                    fig\_treemap.update\_traces(

                        # District names (parent level) - slightly bolder

                        outsidetextfont=dict(

                            size=10,

                            family=CHART\_FONT,

                            color='black'

                        ),

                        # Incident types (nested level) - normal weight

                        insidetextfont=dict(

                            size=9,

                            family=CHART\_FONT,

                            color='black'

                        )

                    )

                    fig\_treemap.update\_layout(

                        title\_text='Incident Distribution by District & Type',

                        margin=dict(t=40, l=5, r=5, b=5),

                        height=450,

                        plot\_bgcolor='white',

                        paper\_bgcolor='white'

                    )

                    st.plotly\_chart(style\_plotly\_chart(fig\_treemap, chart\_height=380, is\_pie\_or\_donut=True), use\_container\_width=True, key=f"treemap\_chart\_{total\_incidents}\_{start\_date}\_{end\_date}")

                except Exception as e:

                    st.error(f"Could not render Treemap: {e}")

            else:

                st.info("No data available to display in the Treemap for the current filter selection.")

        else:

            st.info("Required columns ('district', 'incident\_type') not available or no data in filtered set for Treemap.")

    # Footer

    st.markdown(f'''<div class="footer-card" style="margin-top: 2rem;">

        <h4 class="footer-title">🚨 Bihar Disaster Management Dashboard</h4>

        <p>Built with Streamlit & Plotly | Last updated: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}</p>

        <p>Data visualization for disaster preparedness and emergency response planning.</p>

        <p style="font-size: 0.7rem; margin-top: 0.4rem;">🏛️ Government of Bihar | Emergency Operations Center</p>

        </div>''', unsafe\_allow\_html=True)

if \_\_name\_\_ == "\_\_main\_\_":

    run()

**12.c. Main Dashboard Integration Script**

This section contains the complete Python source code for the unified dashboard launcher, main.py. The main.py helps to present all the 3 Dashboards named Bihar Cold Wave Dashboard as Dashboard1.py, Bihar Bihar Disaster Incident Dashboard as Dashboard2.py and Bihar Flood Dashboard as Dashboard3.py. In the .streamlit folder each Dashboard has their respective .toml file for the database connection as Dashboard1.toml, Dashboard2.toml and Dashboard3.toml for their respective Scripts.

import streamlit as st

import importlib

import base64

# UI config

st.set\_page\_config(page\_title="Unified Dashboard App", layout="wide")

# Image to base64 conversion function

@st.cache\_data

def get\_image\_as\_base64(path):

    try:

        with open(path, "rb") as image\_file:

            encoded\_string = base64.b64encode(image\_file.read()).decode()

        image\_type = path.split('.')[-1].lower()

        if image\_type in ["jpg", "jpeg"]: return f"data:image/jpeg;base64,{encoded\_string}"

        elif image\_type == "png": return f"data:image/png;base64,{encoded\_string}"

        else: return f"data:image/png;base64,{encoded\_string}"

    except FileNotFoundError:

        st.warning(f"Image file not found at path: '{path}'. Please ensure it is in the correct directory.")

        return "data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAABAAEAAAIBRAA7"

#Dashboard map: name -> module name

dashboards = {

    "Cold Wave Dashboard": "Dashboard1",

    "Incident Dashboard": "Dashboard2",

    "Flood Dashboard": "Dashboard3"

}

# Sidebar logos and selection

with st.sidebar:

    # Displaying logos at the top

    eoc\_logo\_base64 = get\_image\_as\_base64("eoc\_logo.png")

    bihar\_logo\_base64 = get\_image\_as\_base64("bihar\_govt.png")

    st.markdown("""

    <style>

    /\* Making sidebar narrower \*/

    .css-1d391kg, .css-1lcbm7v, .css-1v3fv7u {

        width: 250px !important;

        min-width: 250px !important;

        max-width: 250px !important;

    }

    section[data-testid="stSidebar"] {

        width: 250px !important;

        min-width: 250px !important;

        max-width: 250px !important;

    }

    section[data-testid="stSidebar"] > div {

        width: 250px !important;

        min-width: 250px !important;

        max-width: 250px !important;

    }

    .sidebar-logo {

        text-align: center;

        padding: 0.5rem 0.5rem 1.5rem;

        margin-bottom: 1rem;

    }

    .sidebar-logo img {

        filter: none;

        margin: 0.5rem;

        width: 100px;

        height: auto;

    }

    .sidebar-logo img[alt="Bihar Logo"] {

        width: 110px;

    }

    .bihar-text {

        font-size: 0.8rem !important;

        color: #333a40 !important;

        font-weight: 500 !important;

        text-shadow: none !important;

    }

    </style>

    """, unsafe\_allow\_html=True)

    # Displaying the logos

    logo\_html = f'''

    <div class="sidebar-logo">

        <img src="{eoc\_logo\_base64}" alt="EOC Logo"><br>

        <img src="{bihar\_logo\_base64}" alt="Bihar Logo">

        <div class="bihar-text">बिहार सरकार</div>

    </div>

    '''

    st.markdown(logo\_html, unsafe\_allow\_html=True)

selected = st.sidebar.radio("Select a dashboard:", list(dashboards.keys()))

# Displaying info about selected dashboard in sidebar

with st.sidebar:

    st.markdown("---")

    if selected == "Cold Wave Dashboard":

        st.info("🚨 \*\*Cold Wave Dashboard\*\* - Monitor and manage cold wave incidents, affected populations, relief operations, and resource allocation across Bihar districts.")

    elif selected == "Incident Dashboard":

        st.info("🚨 \*\*Incident Dashboard\*\* - Track incidents, casualties, and emergency response operations with detailed analytics and geographical distribution.")

    elif selected == "Flood Dashboard":

        st.info("🚨 \*\*Flood Dashboard\*\* - Comprehensive flood management system with real-time monitoring, district-wise analysis, and disaster response coordination.")

# Importing and running selected dashboard

try:

    dashboard\_module = dashboards[selected]

    dashboard = importlib.import\_module(dashboard\_module)

    dashboard.run()

except Exception as e:

    st.error(f"Failed to run {selected}. Error:\n\n{e}")