

SUMMER INTERNSHIP PROJECT REPORT ON
“DISASTER RELIEF AND RESCUE MANAGEMENT SYSTEM”



बिहार सरकार

Project In-Charge

MR. RAJEEV KUMAR JHA
SCIENTIST-E NATIONAL
INFORMATICS CENTRE
BIHAR STATE CENTRE,
PATNA

Project Mentor

MR. RAKESH KUMAR
SCIENTIST-E NATIONAL
INFORMATICS CENTRE
BIHAR STATE CENTRE,
PATNA

Submitted by

Vedant
B.TECH (CSE)
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
APPLICATION REF. NO. – KIIT-DU/SCE/T&P/584/25
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 VEDANT of BTECH, CSE has presented a project titled "Bihar Flood Relief Dashboard" under the Digital India Summer Internship program during the period 7th May 2025 to 27th June 2025 at the Digital Government Research Centre, Patna.

Project In-charge

Rajiv Kumar Jha

Scientist E

National Informatics Centre

Bihar State Centre

Patna

Project Mentor

Rakesh Kumar

Scientist E

National Informatics Centre

Bihar State Centre

Patna

DECLARATION

I hereby declare that the project report entitled "Bihar Flood Relief 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

The "Bihar Flood Relief Dashboard" is a web-based data visualization tool designed to monitor and analyze the impact of floods across the state of Bihar. Floods are a recurrent and devastating natural disaster in Bihar, causing significant loss of life, property, and livelihood. This project aims to provide a centralized, interactive platform for government officials and disaster management authorities to access real-time and historical flood-related data, enabling better decision-making, resource allocation, and response planning.

The dashboard is developed using Python with the Streamlit framework for the user interface, Pandas for data manipulation, and Plotly for creating rich, interactive visualizations. It presents key performance indicators (KPIs) such as population affected, human and animal loss, and damage to infrastructure. The core feature is an interactive map of Bihar, which visualizes affected districts, alongside dynamic bar charts, trend graphs, and data tables that allow for detailed, district-level analysis.

This project enhances the efficiency and transparency of disaster management operations by consolidating disparate data into a single, easy-to-understand visual format. It serves as a powerful tool for monitoring the on-ground situation and strategizing relief efforts.

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 is one of India's most flood-prone states, with a significant portion of its landmass subject to annual inundation from rivers like the Ganga, Kosi, and Gandak. These floods result in immense loss of life, displacement of communities, damage to agriculture and infrastructure, and a severe impact on the state's economy.

The management of such large-scale disasters requires timely and accurate information. Traditionally, data related to flood impact—such as affected population, relief centers, and damages—is collected and compiled through manual reports from various districts. This process can be slow, prone to errors,

and makes it difficult for state-level decision-makers to get a clear, real-time, consolidated view of the situation. The lack of a centralized, interactive dashboard hinders efficient resource mobilization and strategic planning for relief and rescue operations.

2.2 About the Project

The Bihar Flood Relief Dashboard addresses these challenges by providing a comprehensive, web-based platform for visualizing and analyzing flood-related data. Built with modern data science tools, the dashboard connects to a centralized database to fetch and display critical information in an intuitive and interactive format. This tool aims to empower the Emergency Operations Center and other government bodies with the data-driven insights needed for effective and timely disaster response.

CHAPTER 3: PROJECT OBJECTIVE

The primary objective of this project is to design and develop an interactive and user-friendly dashboard for monitoring flood situations in Bihar. The specific goals are:

- To create a single source of truth by centralizing flood-related data from various districts.
- To provide real-time or near-real-time visualization of Key Performance Indicators (KPIs) related to flood impact.
- To develop an interactive map that visually represents the extent of the disaster across the state.
- To enable comparative analysis between districts to aid in prioritizing resource allocation.
- To build a robust and scalable application using Python and the Streamlit framework that can be easily maintained and enhanced in the future.
- To improve the overall efficiency, transparency, and effectiveness of the disaster management process in the state.

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. It is responsible for rendering the interactive map of Bihar, the district-wise bar charts, and the daily trend graphs, providing features like hover tooltips and zoom.
- **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 FloodMain and FloodDetailsCum.
- **Connection:** The pyodbc library handles the connection to the SQL Server database. Connection details are securely stored in a secrets.toml file.
- **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.

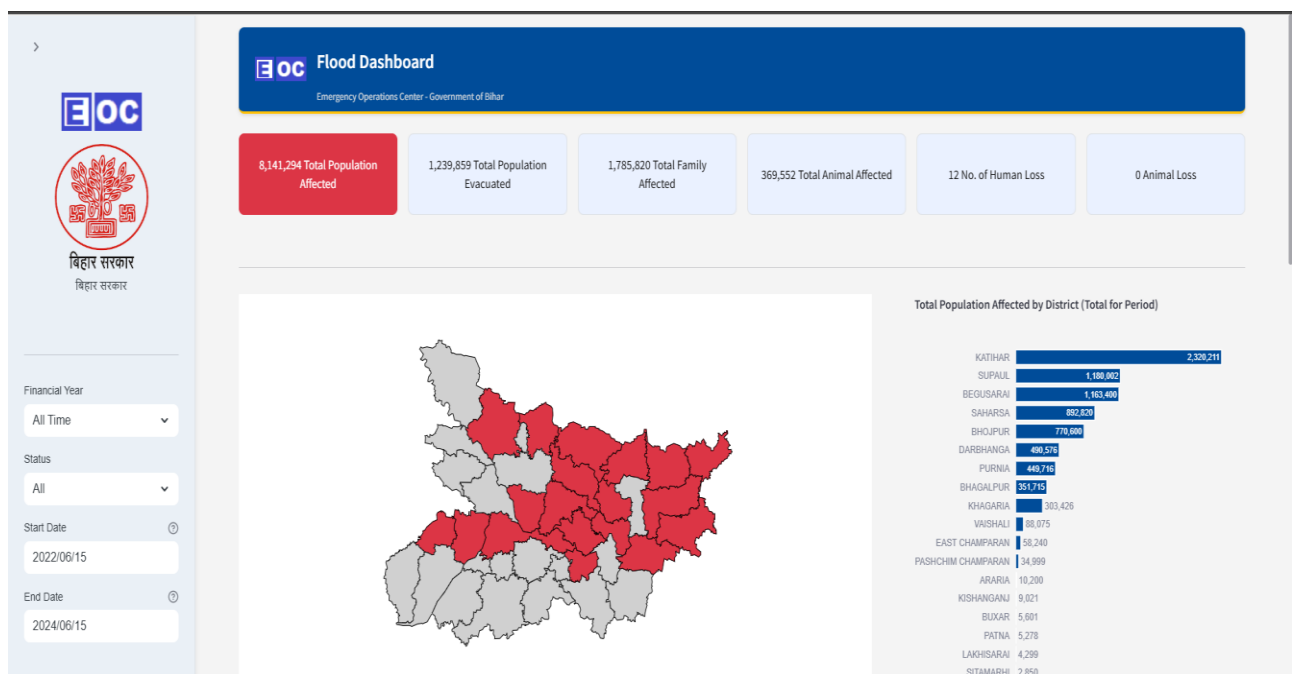
- **Styling:** Custom CSS is injected to create a polished and professional look.

CHAPTER 6: PROJECT FEATURES AND WORKING

The dashboard provides a multi-faceted view of the flood situation through its various interactive components.

6.1 Main Dashboard Layout


Upon launching the application, the user is presented with a clean interface. A header displays the project title. The main content area is divided into sections for KPIs, the map and ranked list, and a detailed drill-down section.



6.2 Sidebar and Filtering

A sidebar on the left contains all user controls for filtering.

- **Time-Based Filters:** Dropdowns and date inputs allow users to select a Financial Year or a custom date range.
- **Report Navigation:** A radio button group allows users to switch between different report categories like "Population Report," "Damage Report," etc., which dynamically changes the available KPIs.



The sidebar on the left of the mobile app contains the following elements:

- Navigation arrows at the top.
- EOC logo.
- Bihar State Emblem (Ashoka Chakra) with the text "बिहार सरकार" (Bihar Government) and "बिहार सरकार" (Bihar Government) below it.
- Financial Year filter: A dropdown menu currently showing "All Time".
- Status filter: A dropdown menu currently showing "All".
- Start Date filter: A date input field showing "2022/06/15".
- End Date filter: A date input field showing "2024/06/15".



The report navigation sidebar on the right contains the following elements:

- A blue button labeled "Population Report".
- Text labels for "Damage Report", "Affected Area Report", "Facility Centre Report", "Relief Report", and "Public Property Damage".

6.3 KPI Card Navigation and Display

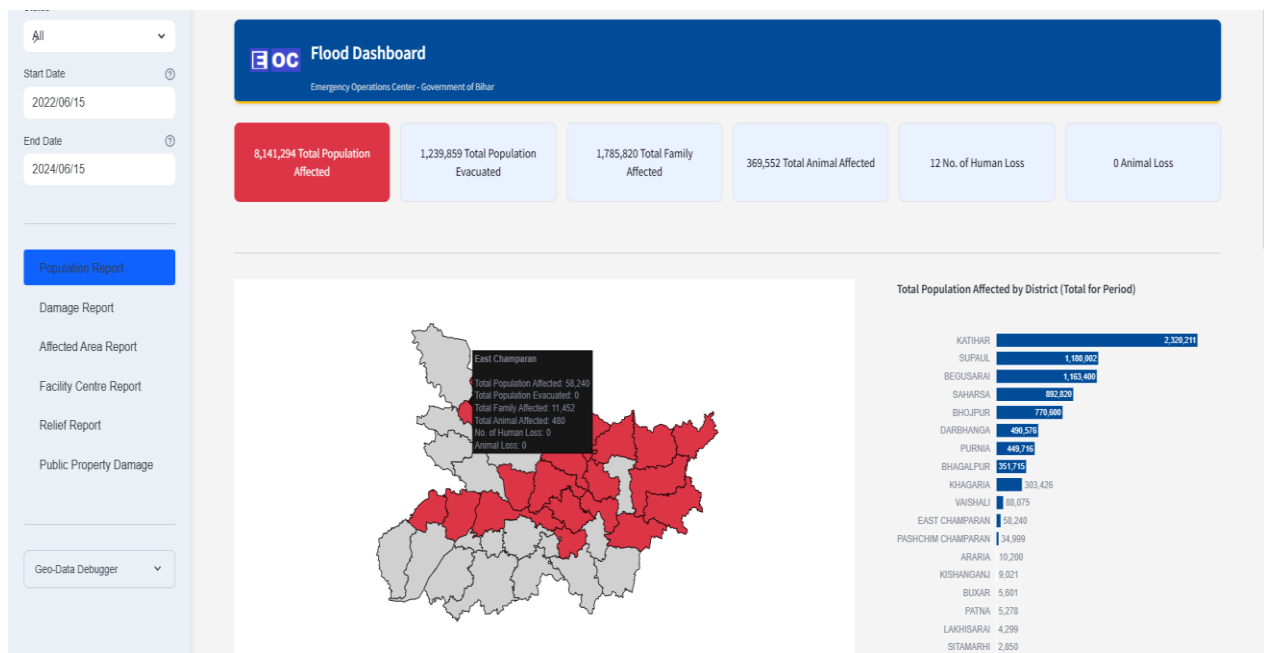
Below the header, a row of cards displays critical KPIs.

- **Dynamic Content:** The KPIs shown change based on the report category selected.
- **Interactivity:** Each KPI card is a clickable button. Clicking a card makes it the primary metric for all other visualizations on the page.

6.4 Interactive Geo-Map

The central feature is an interactive map of Bihar.

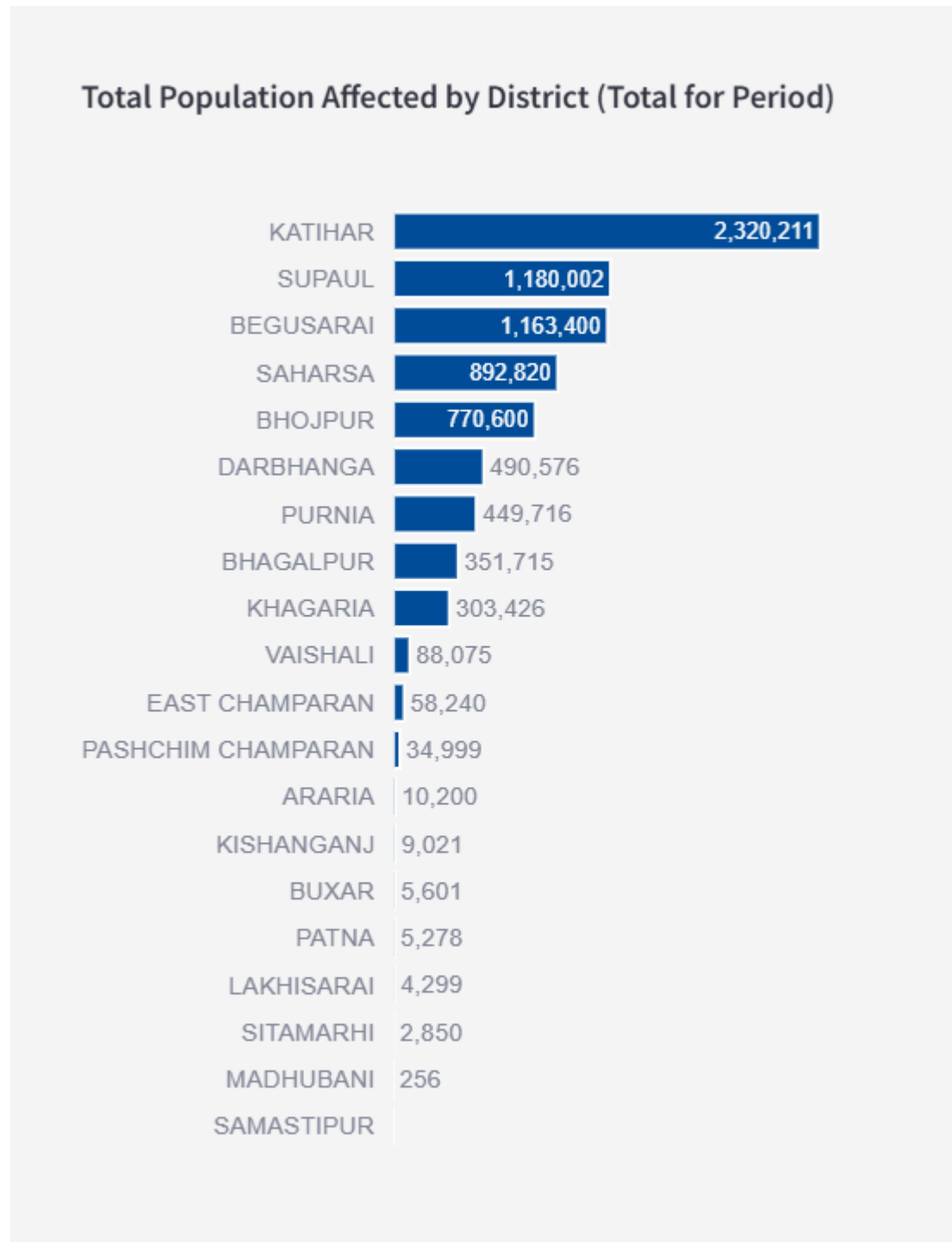
- **Color-Coding:** Districts are colored red if affected according to the selected KPI, and grey otherwise.
- **Hover Tooltip:** Hovering over a district reveals a tooltip with detailed statistics for that specific district.



6.5 District-wise Data Visualization

To the right of the map, a horizontal bar chart provides a ranked list of districts.

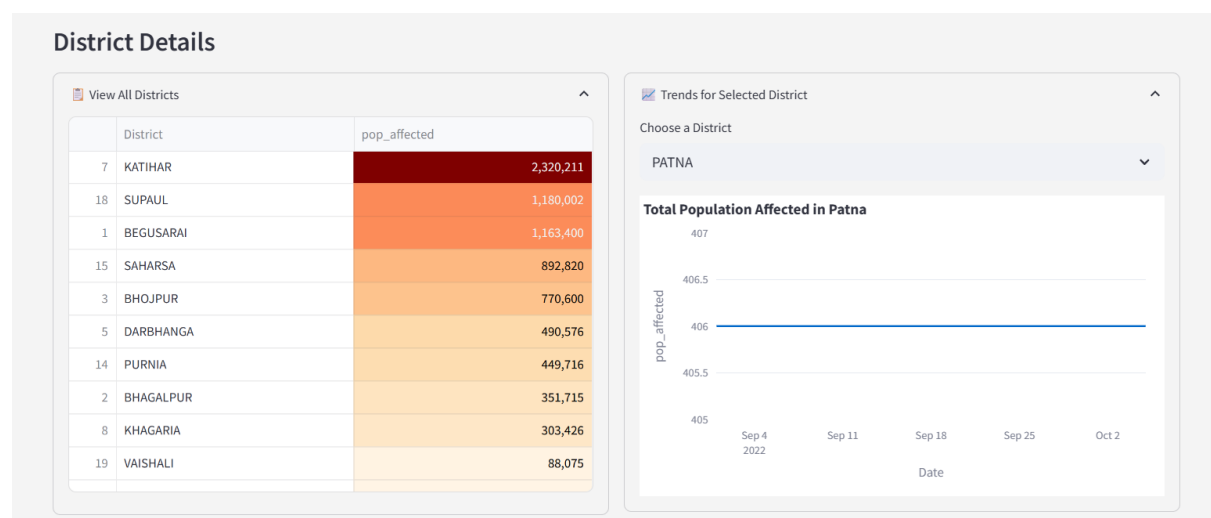
- **Ranking:** The chart displays all districts, sorted in descending order based on their value for the active KPI.



6.6 Drill-Down and Trend Analysis

The lower section of the dashboard allows for deeper analysis.

- **District Details Table:** An expandable section contains a scrollable table listing all districts and their values.
- **Individual Trend Chart:** Users can select a single district to view a line chart of its daily trend for the selected KPI.
- **Aggregate State-level Trends:** Two summary line charts show the daily trend of "Total Affected Districts" and "Total Persons in Relief".



CHAPTER 7: ADVANTAGES

- **Centralized Information:** Provides a single platform for all flood-related data.
- **Improved Situational Awareness:** Offers a quick, visual understanding of the disaster's impact.
- **Data-Driven Decision Making:** Enables authorities to make informed decisions based on real data.
- **Efficient Resource Allocation:** Helps in identifying the worst-hit areas to prioritize relief.
- **Enhanced Transparency:** Makes flood impact data easily accessible to all relevant stakeholder.

- **User-Friendly:** The interactive interface is intuitive and requires no technical expertise.

CHAPTER 8: LIMITATIONS

- **Data Dependency:** The dashboard's accuracy depends on the accuracy and timeliness of data entry.
- **Real-time Constraint:** The dashboard is near-real-time, limited by the database refresh cycle.
- **Scope:** The project is focused on post-disaster analysis and does not include predictive features.

CHAPTER 9: CONCLUSION

The Bihar Flood Relief Dashboard project successfully demonstrates the power of data visualization in disaster management. By transforming raw data into interactive charts, maps, and KPIs, the application provides a clear and actionable overview of the flood situation in Bihar. It addresses the critical need for a centralized monitoring system and serves as a valuable asset for the Emergency Operations Center, aiming to improve the speed and efficiency of relief efforts.

CHAPTER 10: FUTURE SCOPE

- **Predictive Analytics:** Incorporating ML models to predict flood-prone areas.
- **Mobile Application:** Developing a mobile version for officials in the field.
- **Resource Management Module:** Adding a module to track the movement of relief materials.
- **Integration with Other Systems:** Linking the dashboard with early warning systems.

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: APPENDIX

12.1 Full Source Code

This section contains the complete Python source code for the main application file, `nic_dashboard1.py`.

```
import streamlit as st
import pandas as pd
import plotly.graph_objects as go
import plotly.express as px
import json
from datetime import datetime, date, timedelta
import numpy as np
import base64
import pyodbc

# --- 0. Page Configuration ---
st.set_page_config(
    layout="wide",
    page_title="Flood Dashboard - Bihar",
    page_icon="🏠",
    initial_sidebar_state="expanded"
)

# --- KPI Definitions ---
kpi_metric_mapping = {
    "pop_affected": "Total Population Affected", "pop_evacuated": "Total Population Evacuated",
```

```

    "family_affected": "Total Family Affected", "animal_affected_total": "Total
Animal Affected",

    "human_loss": "No. of Human Loss", "animal_loss": "Animal Loss",

    "total_house_damage": "Total House Damage", "kutcha_house_damage_dr":
    "Kutcha House Damage",

    "pucca_house_damage_dr": "Pucca House Damage", "huts_damage_dr":
    "Huts Damage",

    "cost_damage_house_dr": "Est Cost Of Damage House",

    "total_affected_area": "Total Affected Area (Hec.)", "agriculture_area":
    "Agriculture Area (Hec.)",

    "non_agriculture_area": "Non Agriculture Area (Hec.)",
    "crop_damage_area": "Crop Damage Area (Hec.)",

    "damage_fisheries": "Damage Fisheries (Hec.)", "gr_distribution": "Families
GR Distribution",

    "polythene_sheet": "Polythene Sheet", "food_packet": "Food Packet",

    "dry_ration_packet": "Dry Ration Packet", "fodder_distribution": "Fodder
Distribution",

    "est_cost_property_damage": "Est Cost Of Property Damage",

    "fc_boats_total": "Total Boats Deployed",

    "fc_relief_centres_total": "Total Relief Centres",

    "fc_persons_in_relief_total": "Total Persons in Relief",

    "fc_comm_kitchens_total": "Total Community Kitchens",

    "fc_meals_served_total": "Total Meals Served",

    "fc_health_centres_total": "Total Health Centres",

    "fc_persons_treated_total": "Total Persons Treated (Health)",

    "fc_vet_centres_total": "Total Veterinary Centres",

    "fc_animals_treated_total": "Total Animals Treated (Vet)"
}

default_kpi_key = "pop_affected"

default_kpi_label = kpi_metric_mapping[default_kpi_key]

```

```

kpi_options_for_menu = {
    "Population Report": [
        ("Total Population Affected", "pop_affected"), ("Total Population
Evacuated", "pop_evacuated"),
        ("Total Family Affected", "family_affected"), ("Total Animal Affected",
"animal_affected_total"),
        ("No. of Human Loss", "human_loss"), ("Animal Loss", "animal_loss"),
    ],
    "Damage Report": [
        ("Total House Damage", "total_house_damage"), ("Kutch House
Damage", "kutch_house_damage_dr"),
        ("Pucca House Damage", "pucca_house_damage_dr"), ("Huts Damage",
"huts_damage_dr"),
        ("Est Cost Of Damage House", "cost_damage_house_dr"),
    ],
    "Affected Area Report": [
        ("Total Affected Area (Hec.)", "total_affected_area"), ("Agriculture Area
(Hec.)", "agriculture_area"),
        ("Non Agriculture Area (Hec.)", "non_agriculture_area"), ("Crop Damage
Area (Hec.)", "crop_damage_area"),
        ("Damage Fisheries (Hec.)", "damage_fisheries"),
    ],
    "Facility Centre Report": [
        ("Total Boats Deployed", "fc_boats_total"),
        ("Total Relief Centres", "fc_relief_centres_total"),
        ("Total Persons in Relief", "fc_persons_in_relief_total"),
        ("Total Community Kitchens", "fc_comm_kitchens_total"),
        ("Total Meals Served", "fc_meals_served_total"),
    ]
}

```

```

        ("Total Health Centres", "fc_health_centres_total"),
        ("Persons Treated (Health)", "fc_persons_treated_total"),
        ("Total Veterinary Centres", "fc_vet_centres_total"),
        ("Animals Treated (Vet)", "fc_animals_treated_total")
    ],
    "Relief Report": [
        ("Families GR Distribution", "gr_distribution"), ("Polythene Sheet",
"polythene_sheet"),
        ("Food Packet", "food_packet"), ("Dry Ration Packet",
"dry_ration_packet"),
        ("Fodder Distribution", "fodder_distribution"),
    ],
    "Public Property Damage": [("Est Cost Of Property Damage",
"est_cost_property_damage"),],
}
menu_items = list(kpi_options_for_menu.keys())

```

--- Session State Initialization ---

```

if 'selected_menu_memory' not in st.session_state:
    st.session_state.selected_menu_memory = menu_items[0]
if 'selected_main_kpi_key' not in st.session_state:
    initial_kpis_for_default_menu =
kpi_options_for_menu.get(st.session_state.selected_menu_memory, [])
    if initial_kpis_for_default_menu:
        st.session_state.selected_main_kpi_key =
initial_kpis_for_default_menu[0][1]
    else:
        st.session_state.selected_main_kpi_key = default_kpi_key

```

```

if 'district_list_metric_key' not in st.session_state:
    st.session_state.district_list_metric_key =
st.session_state.selected_main_kpi_key

if 'district_list_metric_label' not in st.session_state:
    st.session_state.district_list_metric_label =
kpi_metric_mapping.get(st.session_state.selected_main_kpi_key,
default_kpi_label)

```

--- HELPER FUNCTIONS ---

@st.cache_resource

def init_db_connection():

```

    """Establishes a connection to the SQL Server database using secrets."""
    try:
        conn_str_parts = [
            f"DRIVER={st.secrets.get('db_driver', '{ODBC Driver 17 for SQL Server}')}",
            f"SERVER={st.secrets['db_server']}",
            f"DATABASE={st.secrets['db_database']}",
            "Trusted_Connection=yes",
            "TrustServerCertificate=yes"
        ]
        conn_str = ";".join(conn_str_parts)
        conn = pyodbc.connect(conn_str)
        return conn
    except Exception as e:
        if 'db_server' not in st.secrets:
            return None

```



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

```
return None
```

```
@st.cache_data(ttl=900)
```

```
def load_data_from_db():
```

```
    """Fetches and prepares the main dataset from the SQL database."""
```

```
    conn = init_db_connection()
```

```
    if not conn:
```

```
        st.info("Database connection not available. Loading sample data for  
demonstration.")
```

```
        districts = [
```

```
            "ARARIA", "ARWAL", "AURANGABAD", "BANKA",  
            "BEGUSARAI", "BHAGALPUR",
```

```
            "BHOJPUR", "BUXAR", "DARBHANGA", "GAYA",  
            "GOPALGANJ", "JAHANABAD",
```

```
            "JAMUI", "KAIMUR (BHABUA)", "KATIHAR", "KHAGARIA",  
            "KISHANGANJ",
```

```
            "LAKHISARAI", "MADHEPURA", "MADHUBANI", "MUNGER",  
            "MUZAFFARPUR",
```

```
            "NALANDA", "NAWADA", "PASCHIM CHAMPARAN", "PATNA",  
            "PURBI CHAMPARAN",
```

```
            "PURNIA", "ROHTAS", "SAHARSA", "SAMASTIPUR", "SARAN",  
            "SHEIKHPURA",
```

```
            "SHEOHAR", "SITAMARHI", "SIWAN", "SUPAUL", "VAISHALI"
```

```
        ]
```

```
        data = []
```

```
        start_dt = date.today() - timedelta(days=1000)
```

```
        for i in range(1000):
```

```
            current_date = start_dt + timedelta(days=i)
```

```

for district in districts:
    is_affected = np.random.choice([True, False], p=[0.2, 0.8])
    row = { 'District': district, 'Date': pd.to_datetime(current_date) }
    for key in kpi_metric_mapping.keys():
        row[key] = np.random.randint(0, 10000) if is_affected else 0
    data.append(row)
df_db = pd.DataFrame(data)
else:
    sql_query = """
    SELECT
        dm.DistrictName AS District, m.RecordDate AS Date,
        d.pdHumanAffected AS pop_affected,
        d.pdMigratedPopulation AS pop_evacuated, d.pdFamilyAffected AS
        family_affected,
        d.pdDeadPeoples AS human_loss, d.pdAffectedAnimals AS
        animal_affected_total, 0 AS animal_loss,
        (d.pdPartlyAffectedKutchaHouses + d.pdPartlyAffectedPakkaHouses +
        d.pdAffectedHuts) AS total_house_damage,
        d.pdPartlyAffectedKutchaHouses AS kutcha_house_damage_dr,
        d.pdPartlyAffectedPakkaHouses AS pucca_house_damage_dr,
        d.pdAffectedHuts AS huts_damage_dr, 0 AS cost_damage_house_dr,
        (d.pdAffectedAgriLand + d.pdAffectedNonAgriLand) AS
        total_affected_area, d.pdAffectedAgriLand AS agriculture_area,
        d.pdAffectedNonAgriLand AS non_agriculture_area,
        d.pdDamagedCropArea AS crop_damage_area,
        d.pdDamagedFishSeedFarms AS damage_fisheries,
        d.pdDryRationPackets AS gr_distribution,
        d.pdPolytheneSheetDist AS polythene_sheet, d.pdFoodPackets AS
        food_packet,
        d.pdDryRationPackets AS dry_ration_packet, d.pdOtherItemsDist AS
        fodder_distribution,

```

```

        d.pdDamagedPublicPropVal AS est_cost_property_damage,
d.pdMotorBoatToday AS fc_boats_total,

        d.pdReliefCentreOpened AS fc_relief_centres_total,
d.pdPeopleRegistered AS fc_persons_in_relief_total,

        0 AS fc_comm_kitchens_total, d.pdPeopleDinner AS
fc_meals_served_total,

        d.pdHealthCampToday AS fc_health_centres_total, d.pdPeopleTreated
AS fc_persons_treated_total,

        d.pdAnimalCamps AS fc_vet_centres_total, d.pdAnimalsTreated AS
fc_animals_treated_total

```

```

FROM dbo.FloodMain AS m JOIN dbo.FloodDetailsCum AS d ON m.ID
= d.ID JOIN dbo.mst_Districts AS dm ON m.DistrictCode = dm.DistrictCode;

```

```

"""

```

```

try:

```

```

    df_db = pd.read_sql(sql_query, conn)

```

```

except Exception as e:

```

```

    st.error(f"Failed to load data from the database table. Check your query.
Error: {e}")

```

```

    return pd.DataFrame()

```

```

finally:

```

```

    if conn:

```

```

        conn.close()

```

```

db_to_geojson_map = {

```

```

    "PURBI CHAMPARAN": "EAST CHAMPARAN",

```

```

    "PASCHIM CHAMPARAN": "WEST CHAMPARAN"

```

```

}

```

```

df_db['District'] = df_db['District'].str.strip().str.upper()

```

```

df_db['District'].replace(db_to_geojson_map, inplace=True)

```

```

if 'Date' in df_db.columns:
    df_db['Date'] = pd.to_datetime(df_db['Date'], errors='coerce')
else:
    df_db['Date'] = pd.to_datetime(date.today())
for col in df_db.columns:
    if col not in ['Date', 'District']:
        df_db[col] = pd.to_numeric(df_db[col], errors='coerce').fillna(0)

district_coords = {
    "ARARIA": {"lat": 26.15, "lon": 87.51}, "ARWAL": {"lat": 25.24, "lon":
84.67},
    "AURANGABAD": {"lat": 24.75, "lon": 84.37}, "BANKA": {"lat": 24.88,
"lon": 86.92},
    "BEGUSARAI": {"lat": 25.42, "lon": 86.13}, "BHAGALPUR": {"lat":
25.24, "lon": 86.98},
    "BHOJPUR": {"lat": 25.56, "lon": 84.66}, "BUXAR": {"lat": 25.56, "lon":
83.97},
    "DARBHANGA": {"lat": 26.16, "lon": 85.90}, "GAYA": {"lat": 24.79,
"lon": 85.00},
    "GOPALGANJ": {"lat": 26.46, "lon": 84.43}, "JAHANABAD": {"lat":
25.21, "lon": 84.98},
    "JAMUI": {"lat": 24.92, "lon": 86.22}, "KAIMUR (BHABUA)": {"lat":
25.04, "lon": 83.61},
    "KATIHAR": {"lat": 25.54, "lon": 87.58}, "KHAGARIA": {"lat": 25.50,
"lon": 86.47},
    "KISHANGANJ": {"lat": 26.10, "lon": 87.93}, "LAKHISARAI": {"lat":
25.17, "lon": 86.09},
    "MADHEPURA": {"lat": 25.92, "lon": 86.78}, "MADHUBANI": {"lat":
26.36, "lon": 86.07},
    "MUNGER": {"lat": 25.37, "lon": 86.47}, "MUZAFFARPUR": {"lat":
26.12, "lon": 85.36},

```

```
"NALANDA": {"lat": 25.13, "lon": 85.51}, "NAWADA": {"lat": 24.88, "lon": 85.53},
```

```
"WEST CHAMPARAN": {"lat": 27.16, "lon": 84.35}, "PATNA": {"lat": 25.59, "lon": 85.13},
```

```
"EAST CHAMPARAN": {"lat": 26.66, "lon": 84.91}, "PURNIA": {"lat": 25.77, "lon": 87.47},
```

```
"ROHTAS": {"lat": 25.05, "lon": 84.01}, "SAHARSA": {"lat": 25.88, "lon": 86.60},
```

```
"SAMASTIPUR": {"lat": 25.86, "lon": 85.78}, "SARAN": {"lat": 25.89, "lon": 84.86},
```

```
"SHEIKHPURA": {"lat": 25.13, "lon": 85.85}, "SHEOHAR": {"lat": 26.51, "lon": 85.30},
```

```
"SITAMARHI": {"lat": 26.59, "lon": 85.48}, "SIWAN": {"lat": 26.22, "lon": 84.36},
```

```
"SUPAUL": {"lat": 26.12, "lon": 86.61}, "VAISHALI": {"lat": 25.98, "lon": 85.21},
```

```
}
```

```
df_db['Latitude'] = df_db['District'].str.upper().map(lambda name: district_coords.get(str(name).strip().upper(), {}).get('lat'))
```

```
df_db['Longitude'] = df_db['District'].str.upper().map(lambda name: district_coords.get(str(name).strip().upper(), {}).get('lon'))
```

```
return df_db
```

```
def get_kpi_value(filtered_df, kpi_key, default_val=0):
```

```
    if not filtered_df.empty and kpi_key in filtered_df.columns:
```

```
        try:
```

```
            total = filtered_df[kpi_key].sum()
```

```
            if pd.notna(total) and total == int(total): return int(total)
```

```
            elif pd.notna(total): return round(total, 2)
```

```
        else: return default_val
```

```

except Exception as e:
    print(f"Error calculating KPI for {kpi_key}: {e}")
    return default_val
return default_val

@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///yH5BAEAAAAAL
        AAAAAABAAEAAAIBRAA7"

def generate_fy_list(df):
    if df.empty or 'Date' not in df.columns:
        return ["FY 2025-26", "FY 2024-25"]

    years = df['Date'].dt.year.unique()
    min_year, max_year = min(years), max(years)

```

```

fy_list = ["All Time"]
for year in range(max_year, min_year - 1, -1):
    fy_list.append(f"FY {year}-{year+1}")
return fy_list

def create_donut_chart_display(target_column, affected_count, total_count,
label_text, color):
    actual_affected = max(0, affected_count)
    not_affected = max(0, total_count - actual_affected)
    fig_donut = go.Figure(data=[go.Pie(
        labels=["Affected", "Unaffected"], values=[actual_affected, not_affected],
hole=.7,
        marker_colors=[color, 'rgba(0,0,0,0.05)'],
        textinfo='none', hoverinfo='label+value', sort=False, direction='clockwise'
    )])
    fig_donut.update_layout(
        annotations=[
            dict(text=f"{ actual_affected:}", x=0.5, y=0.55, font_size=14,
showarrow=False, font_weight='bold', font_color=color),
            dict(text="Affected", x=0.5, y=0.40, font_size=7, showarrow=False,
font_color=color, opacity=0.8)
        ], showlegend=False, margin=dict(t=5, b=5, l=5, r=5), height=120,
        paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
        font_family="IBM Plex Sans, sans-serif"
    )
    target_column.plotly_chart(fig_donut, use_container_width=True,
config={'displayModeBar': False})
    target_column.markdown(

```

```

f'"""<div style='text-align:center; margin-top:-20px; margin-bottom:
15px;'>

    <strong style='font-size:0.75em;'>{label_text}</strong><br>

    <span style='font-size:0.65em; opacity:0.7;'>Unaff: {not_affected:,} /
Tot: {total_count:,}</span>

</div>""", unsafe_allow_html=True

)

```

--- Custom CSS and JS ---

```
css_style_block = """
```

```

<style>

.stApp {
    background-color: #F4F4F4 !important;
    margin: 0 !important;
    padding: 0 !important;
    overflow-x: hidden !important;
    max-width: 100vw !important;
}

.block-container {
    max-width: 95% !important;
    padding: 0rem 1.5rem 1.5rem 1.5rem !important;
}

:root {
    --primary-color: #0F62FE;
    --secondary-bg-color: #FFFFFF;
    --app-bg-color: #F4F4F4;
    --text-color: #161616;
    --sidebar-bg-color: #EAF0F6;

```



```
--sidebar-text-color: #333a40;
--card-border-color: #dee2e6;
--card-shadow: 0 2px 4px rgba(0,0,0,0.05);
--kpi-card-light-blue-bg: #EAF1FF;
--custom-kpi-card-selected-bg: #dc3545;
}
```

```
[data-testid="stHeader"], [data-testid="stToolbar"] { display: none
!important; }
```

```
[data-testid="stSidebar"] {
  background-color: var(--sidebar-bg-color) !important;
  min-width: 280px !important; max-width: 280px !important;
  border-right: 1px solid var(--card-border-color) !important;
  box-shadow: 2px 0 5px rgba(0,0,0,0.05) !important; z-index: 1000;
}
```

```
[data-testid="stSidebar"] > div { background-color: var(--sidebar-bg-color)
!important; }
```

```
[data-testid="stSidebar"] * { color: var(--sidebar-text-color) !important; font-
family: 'IBM Plex Sans', sans-serif !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: 120px; height: auto;}
```

```
.sidebar-logo img[alt="Bihar Logo"] { width: 140px; }
```

```
.bihar-text { font-size: 0.9rem !important; color: var(--sidebar-text-color)
!important; font-weight: 500 !important; text-shadow: none !important; }
```

```
[data-testid="stSidebar"] .stRadio > div > label {
  background: transparent !important; color: var(--sidebar-text-color)
!important;
  border-radius: 4px !important; padding: 0.6rem 1rem !important;
  margin-bottom: 0.3rem !important; font-weight: 400 !important;
  font-size: 0.9rem !important; border: none !important; width: 100%
!important;
  transition: all 0.2s ease !important;
  display: flex !important; align-items: center; justify-content: flex-start;
}
```

```
[data-testid="stSidebar"] .stRadio > div > label > div:first-child { display:
none !important; }
```

```
[data-testid="stSidebar"] .stRadio > div > label:has(input:checked) {
  background-color: var(--primary-color) !important;
  color: var(--secondary-bg-color) !important;
  font-weight: 500 !important;
}
```

```
[data-testid="stSidebar"] .stRadio > div >
label:hover:not(:has(input:checked)) {
  background-color: rgba(0,0,0,0.05) !important;
  color: var(--primary-color) !important;
}
```

```
.dashboard-header {
  background-color: #004C99; color: var(--secondary-bg-color);
  padding: 0.5rem 1.5rem; text-align: left; display: flex;
```

```
    align-items: center; border-bottom: 3px solid #ffc107;
    border-radius: 8px;
    box-shadow: 0 2px 4px rgba(0,0,0,0.1);
}

.dashboard-header .header-logo-img { height: 30px; margin-right: 15px; }
.dashboard-header .header-title-block { flex-grow: 1; }
.dashboard-header h1 {
    font-size: 1.5rem; font-weight: 600; margin: 0 0 0.1rem 0;
    line-height: 1.2; color: var(--secondary-bg-color);
}

.dashboard-header p.tagline { font-size: 0.8rem; margin: 0; color: #e0e0e0;
opacity: 0.9; }
```

```
.custom-kpi-card-row-container { margin-top: 1.5rem; margin-bottom:
1.5rem; padding: 0 0.1rem; }
```

```
[data-testid="stButton"] > button {
    padding: 1rem !important;
    border-radius: 8px !important;
    text-align: center !important;
    width: 100% !important;
    min-height: 100px;
    height: 100%;
    display: flex !important;
    flex-direction: column !important;
    justify-content: center !important;
    align-items: center !important;
    transition: all 0.2s ease-in-out;
```

```
    line-height: 1.4 !important;
    font-size: 0.9rem !important;
    font-weight: 500;
}
```

```
[data-testid="stButton"] > button:hover {
    box-shadow: 0 4px 10px rgba(0,0,0,0.1) !important;
    transform: scale(1.03);
}
[data-testid="stButton"] > button:focus {
    box-shadow: 0 0 0 2px var(--primary-color) !important;
}
```

```
[data-testid="stButton"] > button[kind="secondary"] {
    background-color: var(--kpi-card-light-blue-bg) !important;
    border: 1px solid var(--card-border-color) !important;
    color: var(--text-color) !important;
}
```

```
[data-testid="stButton"] > button[kind="secondary"]:hover {
    border-color: var(--primary-color) !important;
}
```

```
[data-testid="stButton"] > button[kind="primary"] {
    background-color: var(--custom-kpi-card-selected-bg) !important;
    border: 1px solid var(--custom-kpi-card-selected-bg) !important;
    color: var(--secondary-bg-color) !important;
}
```

```
}
```

```
[data-testid="stButton"] > button[kind="primary"]:hover {  
    background-color: #b02a37 !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>
```

```
"""
```

```
st.markdown(css_style_block, unsafe_allow_html=True)
```

```
# === APPLICATION START: Load Data ===
```

```
df_main = load_data_from_db()
```

```
if df_main.empty:
```

```
    st.error("Data could not be loaded or the source is empty. Dashboard cannot  
    proceed.")
```

```
    st.stop()
```

```
# --- Initial Filter Values ---
```

```
initial_start_date = df_main["Date"].min().date() if not df_main["Date"].empty  
else date.today() - timedelta(days=30)
```

```
initial_end_date = df_main["Date"].max().date() if not df_main["Date"].empty  
else date.today()
```

```

if 'start_date_main_val' not in st.session_state:
    st.session_state.start_date_main_val = initial_start_date
if 'end_date_main_val' not in st.session_state:
    st.session_state.end_date_main_val = initial_end_date

# --- Sidebar Controls ---
with st.sidebar:
    eoc_logo_base64 = get_image_as_base64("eoc_logo.png")
    bihar_logo_base64 = get_image_as_base64("bihar_govt.png")
    logo_html = f'<div class="sidebar-logo"><br><div class="bihar-text">बिहार सरकार</div></div>'
    st.markdown(logo_html, unsafe_allow_html=True)

    st.markdown("---")
    fy_list = generate_fy_list(df_main)
    selected_fy = st.selectbox(
        "Financial Year", fy_list, key="fy_selector", index=0,
    )

    if selected_fy and selected_fy != st.session_state.get('fy_memory'):
        st.session_state.fy_memory = selected_fy
        if selected_fy == "All Time":
            st.session_state.start_date_main_val = initial_start_date
            st.session_state.end_date_main_val = initial_end_date
        else:
            start_year = int(selected_fy.split(" ")[1].split("-")[0])
            st.session_state.start_date_main_val = date(start_year, 4, 1)

```

```

        st.session_state.end_date_main_val = date(start_year + 1, 3, 31)

    st.rerun()

    status_filter = st.selectbox("Status", ["All", "Affected Only"],
key="status_filter")

    start_date_input = st.date_input(
        "Start Date",
        value=st.session_state.start_date_main_val,
        key="start_date_main_dt",
        help="Select start date for custom range"
    )

    end_date_input = st.date_input(
        "End Date",
        value=st.session_state.end_date_main_val,
        key="end_date_main_dt",
        help="Select end date for custom range"
    )

    if start_date_input != st.session_state.start_date_main_val:
        st.session_state.start_date_main_val = start_date_input
        st.session_state.fy_memory = None
    if end_date_input != st.session_state.end_date_main_val:
        st.session_state.end_date_main_val = end_date_input
        st.session_state.fy_memory = None

    st.markdown("---")

```

```
intended_menu_on_load = st.session_state.get('selected_menu_memory',
menu_items[0])
```

```
try:
```

```
    current_radio_idx = menu_items.index(intended_menu_on_load)
```

```
except ValueError:
```

```
    current_radio_idx = 0
```

```
radio_button_actual_selection = st.radio(
```

```
    "Navigation", menu_items,
```

```
    index=current_radio_idx,
```

```
    key="sidebar_menu_radio_key",
```

```
    label_visibility="collapsed"
```

```
)
```

```
if radio_button_actual_selection !=
```

```
st.session_state.get('selected_menu_memory'):
```

```
    st.session_state.selected_menu_memory = radio_button_actual_selection
```

```
    kpis_for_menu =
```

```
kpi_options_for_menu.get(radio_button_actual_selection, [])
```

```
    if kpis_for_menu:
```

```
        first_kpi_key = kpis_for_menu[0][1]
```

```
        st.session_state.selected_main_kpi_key = first_kpi_key
```

```
        st.session_state.district_list_metric_key = first_kpi_key
```

```
        st.session_state.district_list_metric_label =
```

```
kpi_metric_mapping.get(first_kpi_key)
```

```
    st.rerun()
```

```
st.markdown("---")
```



```

with st.expander("Geo-Data Debugger", expanded=False):
    st.info("This checks for mismatches between database and map file
names.")
    try:
        with open("districts.json", "r", encoding="utf-8") as f:
            geojson_data_debug = json.load(f)

            geojson_districts = {feature['properties']['district'].upper().strip() for
feature in geojson_data_debug['features'] if 'district' in feature.get('properties',
{ })}

            db_districts = {dist.upper().strip() for dist in df_main['District'].unique()
if isinstance(dist, str)}

            st.write(f"Districts in GeoJSON: `{len(geojson_districts)}`")
            st.write(f"Districts in Database: `{len(db_districts)}`")

            db_not_in_geojson = db_districts - geojson_districts
            geojson_not_in_db = geojson_districts - db_districts
            if not db_not_in_geojson and not geojson_not_in_db:
                st.success("✅ All district names match!")
            else:
                if db_not_in_geojson:
                    st.warning("⚠️ DB names NOT in GeoJSON:");
                    st.json(sorted(list(db_not_in_geojson)))

                if geojson_not_in_db:
                    st.warning("⚠️ GeoJSON names NOT in DB:");
                    st.json(sorted(list(geojson_not_in_db)))

            except FileNotFoundError:
                st.error("`districts.json` not found. Cannot perform debug check.")

            except Exception as e:
                st.error(f"An error occurred during the debug check: {e}")

```

```
# --- Main Page Header ---
```

```
eoc_logo_header_base64 = get_image_as_base64("eoc_logo.png")
```

```
header_logo_html = f'<div class="dashboard-header"><div  
class="header-title-block"><h1>Flood Dashboard</h1><p  
class="tagline">Emergency Operations Center - Government of  
Bihar</p></div></div>'
```

```
st.markdown(header_logo_html, unsafe_allow_html=True)
```

```
# --- Data Filtering ---
```

```
df_filtered_by_date = df_main[  
    (df_main["Date"].dt.date >= st.session_state.start_date_main_val) &  
    (df_main["Date"].dt.date <= st.session_state.end_date_main_val)  
>].copy()
```

```
if st.session_state.status_filter == 'Affected Only':
```

```
    primary_kpi_key = st.session_state.get('selected_main_kpi_key',  
    default_kpi_key)
```

```
    affected_districts =  
    df_filtered_by_date[df_filtered_by_date[primary_kpi_key] >  
    0][['District']].unique()
```

```
    df_filtered =  
    df_filtered_by_date[df_filtered_by_date['District'].isin(affected_districts)].copy  
    ()
```

```
else:
```

```
    df_filtered = df_filtered_by_date.copy()
```

```
# --- KPI Cards Display ---
```

```

kpis_for_current_menu =
kpi_options_for_menu.get(st.session_state.selected_menu_memory, [])

def handle_kpi_click(kpi_key):
    st.session_state.selected_main_kpi_key = kpi_key
    st.session_state.district_list_metric_key = kpi_key
    st.session_state.district_list_metric_label = kpi_metric_mapping.get(kpi_key)

st.markdown('<div class="custom-kpi-card-row-container">',
unsafe_allow_html=True)
if kpis_for_current_menu:
    num_kpis = len(kpis_for_current_menu)
    cols_per_row_config = 6
    kpi_idx = 0
    while kpi_idx < num_kpis:
        cols = st.columns(cols_per_row_config)
        for i in range(cols_per_row_config):
            if kpi_idx < num_kpis:
                kpi_label, kpi_key = kpis_for_current_menu[kpi_idx]
                with cols[i]:
                    value = get_kpi_value(df_filtered, kpi_key)
                    value_display = f"{int(value):,}" if isinstance(value, (int, float))
and pd.notna(value) else str(value)
                    is_selected = (kpi_key ==
st.session_state.get('selected_main_kpi_key'))
                    button_label = f"{value_display} {kpi_label}"
                    st.button(
                        label=button_label,
                        key=f"kpi_btn_{kpi_key}",

```

```

        on_click=handle_kpi_click,
        args=(kpi_key,),
        type="primary" if is_selected else "secondary",
        use_container_width=True
    )
    kpi_idx += 1
else:
    st.info(f"No specific summary KPIs to display for
    {st.session_state.selected_menu_memory} in this view.")
    st.markdown('</div>', unsafe_allow_html=True)

# --- Map and Bar Chart ---
st.markdown("---")
map_col, district_list_col = st.columns([0.65, 0.35], gap="large")

with map_col:
    geojson_path = "districts.json"
    try:
        with open(geojson_path, "r", encoding="utf-8") as f:
            geojson_data = json.load(f)
    except Exception as e:
        st.error(f"Could not load GeoJSON file: {e}")
        geojson_data = None

    if geojson_data:
        fig_map = go.Figure()

```

```

numeric_kpi_cols = list(kpi_metric_mapping.keys())

df_filtered['District_Normalized'] =
df_filtered['District'].str.strip().str.upper()

df_district_summary =
df_filtered.groupby('District_Normalized')[numeric_kpi_cols].sum().reset_index()

df_district_summary.set_index('District_Normalized', inplace=True)

kpis_for_hover =
kpi_options_for_menu.get(st.session_state.selected_menu_memory, [])

hovertemplate = "<b>{% text}</b><br><br>" + "<br>".join([f"{label}:"
% { { customdata[{i}]} }" for i, (label, key) in enumerate(kpis_for_hover)]) +
"<extra></extra>"

primary_kpi_key = st.session_state.get('selected_main_kpi_key',
default_kpi_key)

if primary_kpi_key not in df_district_summary.columns:

df_district_summary[primary_kpi_key] = 0

df_district_summary['is_affected'] =
df_district_summary[primary_kpi_key] > 0

for feature in geojson_data["features"]:

district_name = feature["properties"]["district"].strip().upper()

color = "#dc3545" if df_district_summary.get('is_affected',
{ }).get(district_name, False) else "#D0D0D0"

geom = feature.get("geometry", { })

polygons = geom.get("coordinates", [])

if geom.get("type") == "Polygon":

polygons = [polygons]

custom_data_for_district = []

```

```

if district_name in df_district_summary.index:
    district_row = df_district_summary.loc[district_name]
    for _, key in kpis_for_hover:
        custom_data_for_district.append(f"{int(district_row.get(key,
0)):.}"")
    else:
        for _, key in kpis_for_hover:
            custom_data_for_district.append("0")
for poly in polygons:
    if not poly: continue
    exterior_ring = poly[0]
    if not exterior_ring or len(exterior_ring) < 3: continue
    lons, lats = zip(*exterior_ring)

    fig_map.add_trace(go.Scattermapbox(lon=list(lons), lat=list(lats),
mode="lines", fill="toself", fillcolor=color, line=dict(color="black", width=1),
hoverinfo="none", showlegend=False))

    fig_map.add_trace(go.Scattermapbox(lon=list(lons), lat=list(lats),
mode="lines", fill="toself", fillcolor="rgba(0,0,0,0)", line=dict(width=0),
hoverinfo="text", text=[district_name.title()] * len(lons),
customdata=np.array([custom_data_for_district] * len(lons)),
hovertemplate=hovertemplate, hoverlabel=dict(bgcolor="#161616",
font_size=12, bordercolor="black", font_family="IBM Plex Sans, sans-serif"),
showlegend=False))

    fig_map.update_layout(mapbox_style="white-bg", mapbox_center={"lat":
25.78, "lon": 85.77}, mapbox_zoom=6.2, margin={"r":0,"t":0,"l":0,"b":0},
height=520, showlegend=False)

    st.plotly_chart(fig_map, use_container_width=True,
config={'displayModeBar': False})

with district_list_col:

```

```

active_metric_display_label = st.session_state.district_list_metric_label
active_metric_key_for_list = st.session_state.district_list_metric_key
st.markdown(f"**{active_metric_display_label} by District (Total for Period)**")

all_districts = sorted(df_main['District'].unique())

district_data_sum =
df_filtered.groupby("District")[active_metric_key_for_list].sum()

district_data_bar = district_data_sum.reindex(all_districts,
fill_value=0).sort_values(ascending=False)

if not district_data_bar.empty:

    fig_bar = go.Figure(go.Bar(x=district_data_bar.values,
y=district_data_bar.index, orientation='h', text=district_data_bar.apply(lambda
x: f'{x:,.0f}' if x > 0 else ""), textposition='auto', marker_color='#004C99',
marker_line_color='white', marker_line_width=1.5))

    fig_bar.update_layout(height=480, margin=dict(t=25, b=0, l=10, r=20),
yaxis=dict(autorange="reversed"), xaxis=dict(showticklabels=False),
plot_bgcolor='rgba(0,0,0,0)', paper_bgcolor='rgba(0,0,0,0)', font_family="IBM
Plex Sans, sans-serif")

    st.plotly_chart(fig_bar, use_container_width=True,
config={'displayModeBar': False})

else:

    st.info("No affected districts for this metric.")

# --- Drill-down KPI Section ---
st.markdown("---")
st.subheader("District Details")
col1, col2 = st.columns(2)
metric_key = st.session_state.get('selected_main_kpi_key', default_kpi_key)

with col1:

```

```

with st.expander("📋 View All Districts", expanded=True):

    df_table_data =
df_filtered.groupby('District')[metric_key].sum().reset_index()

    df_table = df_table_data[['District', metric_key]].sort_values(metric_key,
ascending=False)

    st.dataframe(df_table.style.format({ metric_key:
":,.0f"})).background_gradient(cmap='OrRd', subset=[metric_key]),
use_container_width=True, height=400)

with col2:

    with st.expander("📈 Trends for Selected District", expanded=True):

        unique_districts = sorted(df_filtered['District'].unique())

        if unique_districts:

            district_choice = st.selectbox("Choose a District", unique_districts)

            if district_choice:

                kpi_label = kpi_metric_mapping.get(metric_key, metric_key)

                df_trend = df_filtered[df_filtered['District'] ==
district_choice].groupby('Date')[metric_key].sum().reset_index()

                fig_trend_line = px.line(df_trend, x='Date', y=metric_key,
title=f"{kpi_label} in {district_choice.title()}")

                fig_trend_line.update_layout(margin=dict(l=20, r=20, t=40, b=20),
height=320)

                st.plotly_chart(fig_trend_line, use_container_width=True)

            else:

                st.info("No districts with data in the selected period.")

# --- Daily Trends and Donut Charts ---

st.markdown("<br>", unsafe_allow_html=True)

TOTAL_DISTRICTS, TOTAL_BLOCKS, TOTAL_NAGARS,
TOTAL_PANCHAYATS = 38, 534, 200, 8386

```



```

affected_districts_count = df_filtered[df_filtered[default_kpi_key] >
0]['District'].nunique() if not df_filtered.empty else 0

affected_blocks_count = int(affected_districts_count * 5) if
affected_districts_count > 0 else 0

affected_panchayats_count = int(affected_blocks_count * 4) if
affected_blocks_count > 0 else 0


donut_cols = st.columns(4)

create_donut_chart_display(donut_cols[0], affected_districts_count,
TOTAL_DISTRICTS, "Districts", '#004C99')

create_donut_chart_display(donut_cols[1], affected_blocks_count,
TOTAL_BLOCKS, "Blocks", '#28a745')

create_donut_chart_display(donut_cols[2], 0, TOTAL_NAGARS, "Nagars",
'#ffc107')

create_donut_chart_display(donut_cols[3], affected_panchayats_count,
TOTAL_PANCHAYATS, "Panchayats", '#dc3545')


trend_graph_col1, trend_graph_col2 = st.columns(2)

s_date_dt = st.session_state.start_date_main_val
e_date_dt = st.session_state.end_date_main_val
all_days_range = pd.date_range(s_date_dt, e_date_dt, freq='D')


with trend_graph_col1:

    st.markdown("##### Daily Affected Districts")

    if not df_filtered.empty:

        daily_counts = df_filtered[df_filtered[default_kpi_key] >
0].groupby(df_filtered['Date'].dt.date)['District'].nunique().reindex(all_days_ran
ge.date, fill_value=0)

    else:

        daily_counts = pd.Series(0, index=all_days_range.date)

```

```

daily_counts.index = pd.to_datetime(daily_counts.index)

fig_trends_districts = go.Figure()

fig_trends_districts.add_trace(go.Scatter(x=daily_counts.index,
y=daily_counts.values, mode='lines', name='Districts Affected',
line=dict(color='#004C99', width=3, shape='spline'), fill='tozeroy',
fillcolor='rgba(0, 76, 153, 0.1)'))

fig_trends_districts.update_layout(height=270, margin=dict(t=30, b=50,
l=60, r=20), paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
xaxis=dict(gridcolor='#E0E0E0', showline=False, zeroline=False),
yaxis=dict(gridcolor='#E0E0E0', showline=False, zeroline=False),
font_family="IBM Plex Sans, sans-serif", font_color="#161616",
yaxis_title="Affected Districts", xaxis_title="Date", hovermode="x unified",
showlegend=False)

st.plotly_chart(fig_trends_districts, use_container_width=True,
config={'displayModeBar': False})

with trend_graph_col2:

    st.markdown("##### Daily Persons in Relief")

    if not df_filtered.empty:

        daily_relief =
df_filtered.groupby(df_filtered['Date'].dt.date)['fc_persons_in_relief_total'].sum
().reindex(all_days_range.date, fill_value=0)

    else:

        daily_relief = pd.Series(0, index=all_days_range.date)

        daily_relief.index = pd.to_datetime(daily_relief.index)

        fig_trends_relief = go.Figure()

        fig_trends_relief.add_trace(go.Scatter(x=daily_relief.index,
y=daily_relief.values, mode='lines', name='Persons in Relief',
line=dict(color='#28a745', width=3, shape='spline'), fill='tozeroy',
fillcolor='rgba(40, 167, 69, 0.05)'))

        fig_trends_relief.update_layout(height=270, margin=dict(t=30, b=50, l=60,
r=20), paper_bgcolor='rgba(0,0,0,0)', plot_bgcolor='rgba(0,0,0,0)',
xaxis=dict(gridcolor='#E0E0E0', showline=False, zeroline=False),

```

```
yaxis=dict(gridcolor='#E0E0E0', showline=False, zeroline=False),
font_family="IBM Plex Sans, sans-serif", font_color="#161616",
yaxis_title="Total Persons in Relief", xaxis_title="Date", hovermode="x
unified", showlegend=False)
```

```
st.plotly_chart(fig_trends_relief, use_container_width=True,
config={'displayModeBar': False})
```

```
# --- 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)
```