ISRO TELEMETRY, TRACKING AND COMMAND NETWORK (ISTRAC)

PROJECT REPORT

NISAR Operations Monitoring System

Submitted By:

Himanshu B A

Bachelor of Engineering (BE),

Information Science & Engineering,

Dayananda Sagar College of Engineering

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute affiliated to Visvesvaraya Technological University (VTU), Belagavi,
Approved by AICTE and UGC, Accredited by NAAC with 'A' grade & ISO 9001-2015 Certified Institution)

Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru - 560 111. India

Under the guidance of,

Mr. Bijoy Kumar Dai

Sci/Eng-SF, SPOA/ISTRAC/ISRO

Bangalore, India

Submitted To:

Mr. B Sankar Madaswamy,

HRD Manager, ISTRAC/ISRO

Bangalore, Indi

ACKNOWLEDGEMENT

I extend my deepest gratitude to **Mr. Bijoy Kumar Dai, Sci/Eng-SF, SPOA, ISTRAC, ISRO** for his invaluable guidance, unwavering support, and the generous time he dedicated to me every day over the past three months. His profound knowledge in space technology and his commitment to the project were instrumental in shaping my research and enriching our understanding. I am sincerely grateful for the opportunity to learn from him and for his continuous dedication to our growth.

My appreciation extends to **Mr. B. Sankar Madaswamy, HR Manager, ISTRAC, ISRO** for facilitating our collaboration with ISRO and providing the necessary resources to ensure the smooth execution of our project.

I would like to extend our heartfelt thanks to **Dr. Annapurna P Patil (Dean Academics , Professor & Head Department of Information Science & Engineering, Dayananda Sagar College of Engineering)**, for her constant support and encouragement in helping me explore new boundaries in my career, as well as for providing the required documents that helped me secure an internship opportunity at ISRO.

Lastly, I extend my gratitude to the entire team at ISTRAC for their warm welcome and cooperation during my time at the organization, and to my family and friends for their unwavering support and encouragement throughout this journey.

ABOUT THE ORGANIZATION

The **Indian Space Research Organisation (ISRO),** over the years, has established a comprehensive global network of ground stations to provide Telemetry, Tracking and Command (TTC) support to satellite and launch vehicle missions. These facilities are grouped under **ISRO Telemetry, Tracking and Command Network (ISTRAC)** with its headquarters at Bangalore, India.

ISRO Telemetry, Tracking and Command Network (ISTRAC), Bengaluru is entrusted with the major responsibility to provide tracking support for all the satellite and launch vehicle missions of **ISRO**. The major objectives of the centre are: carrying out mission operations of all operational remote sensing and scientific satellites, providing Telemetry, Tracking and Command (TTC) services from launch vehicle lift-off till injection of satellite into orbit and to estimate its preliminary orbit in space and hardware and software developmental activities that enhance the capabilities of **ISTRAC** for providing flawless TTC and Mission Operations services. Towards, these objectives, **ISTRAC** has established a network of ground stations at Bengaluru, Lucknow, Mauritius, Sriharikota, Port Blair, Thiruvananthapuram, Brunei, Biak (Indonesia), Bharti Research Station (AGEOS) and the Deep Space Network Stations.

In keeping with its long-established TTC support responsibility, **ISTRAC** has also been mandated to provide space operations support for Deep Space Missions of **ISRO**, undertake development of radar systems for launch vehicle tracking and meteorological applications, establish and operationalise the ground segment for Indian Regional Navigational Satellite System, provide Search & Rescue and Disaster Management Services and support space based services like telemedicine, Village Resource Centre (VRC) and tele-education.

ABSTRACT

The NISAR Operations Monitoring System is an innovative platform designed to enhance the visualization and analysis of data from the NASA-ISRO Synthetic Aperture Radar (NISAR) mission. This system leverages advanced interactive technologies, including Dash and Plotly, to provide mission planners and engineers with a comprehensive tool for monitoring and optimizing satellite operations. By transforming complex datasets into dynamic and engaging visualizations, the system facilitates real-time data exploration and informed decision-making.

The primary objective of the NISAR Operations Monitoring System is to improve operational awareness by offering a detailed overview of the mission's activities. Users can monitor payload operations, communication passes, and eclipse periods with ease, gaining insights into the efficiency and performance of the satellite's instruments. The system's interactive features, such as zooming, hovering, and dynamic updates, allow users to explore data in depth, uncovering trends and anomalies that may not be apparent in static visualizations.

A key strength of the system is its ability to support resource optimization. By analyzing patterns in power and bandwidth usage, users can ensure that resources are allocated effectively, maximizing the scientific return of the mission. The system's user-friendly interface and intuitive design make it accessible to a wide range of users, encouraging frequent interaction and exploration of the data.

Despite its successes, the system also highlights areas for future research and improvement. Incorporating advanced analytical tools, such as machine learning algorithms and predictive modeling, could provide deeper insights into the mission's data. Enhancing real-time data integration and scalability will be essential as the volume of data continues to grow. Additionally, expanding the system's compatibility with different devices and platforms will increase accessibility and ensure that users can access the system from anywhere.

CONTENTS

CONTENTS **Pg No.**

FIGURE OF CONTENTS

FIGURES **Pg No.**

1. Introduction

1.1 Overview of NISAR operation

The NISAR Operations Monitoring System is an advanced tool developed to support the monitoring and management of the NASA-ISRO Synthetic Aperture Radar (NISAR) mission. This mission is a collaborative effort between NASA and the Indian Space Research Organisation (ISRO) to provide critical data for understanding Earth's dynamic processes, including ecosystem disturbances, ice-sheet collapse, and natural hazards such as earthquakes, tsunamis, and landslides.

Purpose:

The primary purpose of the NISAR Operations Monitoring System is to provide mission planners and engineers with a comprehensive and interactive platform to visualize and analyze the mission's operational data. By offering detailed insights into the mission's timeline, the system aids in optimizing operational strategies and ensuring the mission's success.

Features

Interactive Visualization: The system employs interactive dashboards to present operational data in a user-friendly manner. Users can explore different days and view specific events and eclipse periods through intuitive visualizations.

Data Integration: The system integrates data from multiple sources, including payload operation plans, dump operation plans, and TM-TC pass details. This integration ensures that users have access to the most up-to-date and relevant information.

Eclipse Monitoring: A key feature of the system is its ability to track and display eclipse periods, which are critical for planning operations that rely on solar power. The eclipse timeline provides a clear view of when the satellite will be in the Earth's shadow.

Real-Time Updates: Designed to run locally, the system opens a web browser to display the dashboard, providing real-time updates as new data becomes available.

Technical Approach

The system is built using modern web technologies, including Dash and Plotly for the front-end and Pandas for data manipulation. This combination allows for the creation of dynamic and interactive visualizations that are both informative and aesthetically pleasing.

Overall, the NISAR Operations Monitoring System is a vital tool for mission management, offering a detailed and interactive view of the mission's operations and helping to ensure its success.

1.2 Importance of payload function analysis

Payload function analysis is crucial in satellite missions like NISAR, as it directly impacts the mission's ability to achieve its scientific and operational objectives. By analyzing payload functions, mission planners can ensure that the satellite's instruments are operating optimally, collecting high-quality data essential for understanding Earth's dynamic processes. This analysis helps in identifying and resolving potential issues, such as instrument malfunctions or data transmission errors, which could compromise the mission's success. Additionally, it aids in optimizing resource allocation, such as power and bandwidth, ensuring that the satellite operates efficiently even during challenging conditions like eclipses. Ultimately, payload function analysis is vital for maximizing the scientific return of the mission and ensuring its long-term success.

1.3 Objectives of the report

he NISAR Operations Monitoring System is designed with several key objectives in mind to support the mission's success:

2. **Enhance Operational Awareness:** Provide mission planners and engineers with a comprehensive view of the mission's operational timeline, including payload activities and eclipse periods, to facilitate informed decision-making.

3. **Optimize Resource Management**: Enable efficient allocation and management of resources such as power and data bandwidth by providing detailed insights into the timing and duration of various operational events.

4. **Improve Data Quality and Reliability:** Ensure the satellite's instruments are functioning optimally by monitoring payload operations and identifying potential issues that could affect data quality or transmission.

5. **Facilitate Real-Time Monitoring:** Offer real-time updates and interactive visualizations to allow for immediate assessment and response to changing mission conditions, enhancing the ability to adapt to unforeseen challenges.

6. **Support Mission Planning and Analysis:** Provide a robust platform for analyzing past and current mission data, aiding in the planning of future operations and the continuous improvement of mission strategies.

By achieving these objectives, the system aims to maximize the scientific and operational success of the NISAR mission, contributing valuable data for understanding Earth's dynamic processes.

2. Literature Review

2.1 Key Components of NISAR Operation

The NASA-ISRO Synthetic Aperture Radar (NISAR) mission is a groundbreaking collaboration between NASA and the Indian Space Research Organisation (ISRO) aimed at providing critical data for understanding Earth's dynamic processes. The mission's primary objective is to monitor and measure changes in the Earth's surface, including ecosystem disturbances, ice-sheet dynamics, and natural hazards such as earthquakes, tsunamis, and landslides.

- **Synthetic Aperture Radar (SAR) Technology:** NISAR utilizes advanced SAR technology to capture high-resolution images of the Earth's surface. This technology allows for the detection of minute changes in land and ice, providing valuable data for scientific research and disaster response.

- **Dual-Frequency Radar:** The mission features a dual-frequency radar system, operating in both L-band and S-band frequencies. This dual capability enhances the mission's ability to penetrate vegetation and soil, offering comprehensive insights into various environmental conditions.
- **Global Coverage and High Revisit Frequency:** NISAR is designed to provide global coverage with a high revisit frequency, ensuring that changes in the Earth's surface are monitored consistently over time. This capability is crucial for tracking dynamic processes and assessing the impact of natural and human-induced changes.
- **Collaborative Effort:** As a joint mission, NISAR leverages the expertise and resources of both NASA and ISRO. This collaboration enhances the mission's scientific and operational capabilities, enabling the collection and analysis of data that meets the needs of a diverse range of stakeholders.
- **Data Utilization:** The data collected by NISAR is expected to support a wide array of applications, from environmental monitoring and resource management to disaster response and climate change research. The mission's data products will be made available to the global scientific community, fostering collaboration and innovation.

The NISAR Operations Monitoring System plays a critical role in supporting the mission by providing real-time insights into operational events and ensuring the efficient management of resources. Through its advanced visualization and analysis capabilities, the system helps maximize the mission's scientific return and operational success.

2.2 Role of Data visualization in aerospace operations

Data visualization is a pivotal component in aerospace operations, providing a means to interpret complex datasets and facilitate informed decision-making. In the context of satellite missions like NISAR, visualization tools transform raw data into intuitive graphical representations, enabling mission planners and engineers to quickly assess operational status and identify potential issues. By offering real-time insights into parameters such as payload performance, resource allocation, and environmental conditions, data visualization enhances situational awareness and supports proactive management. Furthermore, it aids in communicating complex information to diverse stakeholders, ensuring that all parties have a clear understanding of mission progress and challenges. Ultimately, effective data visualization contributes to the optimization of operations, the mitigation of risks, and the achievement of mission objectives.

2.3 Overview of plotly and Interactive Graphing

Plotly is a powerful open-source graphing library that enables the creation of interactive and visually appealing data visualizations. Widely used in data science and analytics, Plotly supports a variety of chart types, including line plots, bar charts, scatter plots, and 3D graphs, making it versatile for different data visualization needs. One of its key features is interactivity, allowing users to explore data through zooming, panning, and hovering over data points for detailed information. This interactivity is particularly beneficial in aerospace operations, where dynamic data exploration can lead to deeper insights and more informed decision-making. Plotly integrates seamlessly with Python, making it accessible for developers and data scientists to incorporate into their projects. Additionally, Plotly's compatibility with Dash, a web application framework, allows for the development of comprehensive dashboards that combine multiple visualizations, providing a holistic view of complex datasets. This capability is essential for missions like NISAR, where real-time data analysis and visualization are critical for operational success.

3. Dataset Description

The NISAR Operations Monitoring System relies on a comprehensive set of datasets to provide detailed insights into the mission's operations. These datasets are integral to visualizing and analyzing the various aspects of the mission, including payload activities, operational events, and eclipse periods.

Key Datasets:

- **Payload Operation Data:** This dataset includes information on the planned and executed payload operations, detailing the timing, duration, and nature of each activity. It is crucial for monitoring the performance and efficiency of the satellite's instruments.
- **Dump Operation Data:** This dataset captures the data dump operations, which involve the transfer of collected data from the satellite to ground stations. It includes timestamps and event types, helping to ensure that data is transmitted and received as expected.
- **TM-TC Pass Details**: The Telemetry and Telecommand (TM-TC) dataset provides information on the communication passes between the satellite and ground stations. It includes details such as station identifiers, start and end times, and the duration of each pass, which are essential for maintaining effective communication.
- **Eclipse Data:** This dataset outlines the periods when the satellite is in the Earth's shadow, known as eclipses. It includes start and end times, allowing for the planning of operations that depend on solar power availability.

- **Data Format and Sources:**
- The datasets are stored in text files, with each entry containing structured information such as timestamps, event types, and other relevant parameters.
- Data is read and processed using Pandas, a powerful data manipulation library in Python, ensuring efficient handling and analysis.
- The system is designed to update these datasets regularly, providing real-time insights into the mission's operations.

By integrating these datasets, the NISAR Operations Monitoring System offers a comprehensive view of the mission's activities, supporting effective planning, monitoring, and analysis.

3.1 Overview of the Four Datasets used

The NISAR Operations Monitoring System leverages four key datasets, each playing a crucial role in providing a comprehensive view of the mission's operations. Below is a detailed description of each dataset:

1. **Payload Operation Data (data.txt):**
    - **Structure:** This dataset is organized with columns representing the date and time of the operation, the agency responsible, the event type, and additional parameters (a, b, c, d, e) that provide further details about the operation.
    - **Example Entry:** 2024-168T10:30:00 NASA EVENT1 1 2 3 4 5
    - **Purpose:** The dataset is used to track and analyze the performance of the satellite's payload instruments. By monitoring these operations, mission planners can ensure that the instruments are functioning correctly and collecting the necessary data for scientific analysis.

Dump Operation Data (data2.txt):

- **Structure:** This dataset includes columns for the date and time of the dump operation, the agency or system involved, and the event type. It may also include additional parameters specific to the operation.
- **Example Entry:** 2024-168T08:00:00 NASA DGA_READY
- **Purpose:** The dataset is essential for verifying the successful transmission of data from the satellite to ground stations. It helps in ensuring data integrity and identifying any issues in the data transfer process.

TM-TC Pass Details (data3.txt):

- **Structure:** This dataset contains columns for the date and time of the pass, the event type, station identifiers, start and end times of the pass, and additional parameters (d, e, key) that provide context for the communication pass.
- **Example Entry:** 2024-168T11:30:00 168 EVENT STATION1 11:30 13:45 d e TM-TC
- **Purpose:** The dataset is crucial for maintaining effective communication between the satellite and ground stations. It ensures that commands are executed correctly and that data is retrieved efficiently.

Eclipse Data (data5.txt):

- **Structure:** This dataset includes columns for the eclipse number, year, start and end dates, and times (a, b, c for start; a1, b1, c1 for end). It provides a detailed timeline of when the satellite will be in the Earth's shadow.
- **Example Entry:** 1 2024 168 168 10 30 0 2024 169 169 12 45 0
- **Purpose:** The dataset is vital for planning operations that depend on solar power. By knowing when the satellite will be in eclipse, mission planners can adjust operations to ensure continuous functionality and data collection.

Each dataset is processed using Pandas, allowing for efficient data manipulation and analysis. The integration of these datasets into the NISAR Operations Monitoring System enables real-time monitoring and decision-making, ensuring the mission's success and maximizing its scientific return.

3.2 Data collection and preprocessing

The NISAR Operations Monitoring System collects data from various operational logs and telemetry sources, which are stored in structured text files. These files contain detailed records of payload operations, data dumps, TM-TC passes, and eclipse periods. Using the Pandas library, the data is read into dataframes where preprocessing steps are applied. This includes cleaning the data to remove any inconsistencies, handling missing values to ensure completeness, and converting timestamps into a uniform format for consistency. Preprocessing also involves sorting and filtering the data to prepare it for analysis. This ensures that the data is accurate and reliable, providing a solid foundation for generating meaningful visualizations and insights.

3.3 Structure and key features of the data

The datasets used in the system are structured with specific columns that capture essential information about the mission's operations. Key features include timestamps that indicate when events occur, event types that describe the nature of each operation, and additional parameters that provide context and details. This structured format allows for efficient data manipulation, enabling users to filter and sort the data based on various criteria. The

datasets are designed to support the system's interactive features, such as filtering by date or event type, which enhances the user's ability to explore and analyze the data effectively.

4. Methodology

4.1 Data processing and plotting approach

The data processing and plotting approach for the NISAR Operations Monitoring System is designed to ensure that the data is accurately represented and easily interpretable. Initially, the raw data is imported into the system using the Pandas library, which provides powerful tools for data manipulation. The first step involves cleaning the data, where any inconsistencies, such as missing values or erroneous entries, are addressed. This is crucial for maintaining the integrity of the analysis.

Once the data is cleaned, it is sorted and filtered based on relevant criteria, such as date and event type, to focus on the specific aspects of the mission that are of interest. Aggregation techniques may also be applied to summarize data points, allowing for a clearer overview of trends and patterns over time. For instance, operational events can be grouped by day to visualize daily activities effectively.

After preprocessing, the data is prepared for visualization using Plotly, a versatile graphing library that supports a wide range of interactive chart types. The plotting approach emphasizes clarity and interactivity, enabling users to engage with the data dynamically. Various visualizations, such as bar charts for operational events and timelines for eclipse periods, are created to provide a comprehensive view of the mission's activities.

Each visualization is designed to highlight key insights, such as the frequency of specific events or the duration of eclipses. The interactive nature of Plotly allows users to zoom in on specific data points, hover over elements for additional information, and filter the data in real-time. This approach not only enhances the user experience but also facilitates deeper analysis, enabling mission planners and engineers to make informed decisions based on the visualized data. Overall, the data processing and plotting methodology is integral to the success of the NISAR Operations Monitoring System, ensuring that complex datasets are transformed into actionable insights.

4.2 Implementation of graphs

The implementation of graphs in the NISAR Operations Monitoring System involves a systematic approach to ensure that the visualizations are both informative and interactive. Here's a detailed breakdown of the process:

1. **Data Preparation:**

The first step involves reading and preprocessing the data using Pandas. This includes cleaning the data, handling missing values, and converting timestamps into a consistent format. The data is then filtered and sorted based on the specific requirements of the visualizations.

2. **Graph Type Selection:**

- Different types of graphs are chosen based on the nature of the data and the insights required. Common graph types used in the system include:
- **Bar Charts:** To represent operational events, showing the frequency or duration of specific activities.
- **Timelines:** To illustrate eclipse periods and operational timelines, providing a clear view of when specific events occur.

3. **Using Plotly for Visualization:**

The Plotly library is utilized to create the graphs. Plotly's capabilities allow for the creation of interactive visualizations that enhance user engagement. Each graph is defined using Plotly's syntax, specifying the data, layout, and any additional features such as hover text and legends.

4. **Interactivity Features:**

To make the graphs more user-friendly, interactive features are integrated. This includes:

- **Zooming:** Users can zoom in on specific areas of the graph to focus on particular data points.
- **Hovering:** Hovering over data points reveals additional information, such as exact values and descriptions, enhancing the understanding of the data.
- **Dynamic Updates:** The graphs are designed to update dynamically based on user interactions, such as selecting different days or datasets.

5. **Integration into Dash:**

The graphs are embedded into the Dash application, allowing for seamless integration with the overall user interface. Each graph is linked to callback functions that respond to user inputs, ensuring that the visualizations reflect the latest data and user selections.

6. **Testing and Optimization:**

After implementation, the graphs are tested for performance and usability. Feedback from users is gathered to identify areas for improvement, ensuring that the visualizations meet the needs of mission planners and engineers.

4.3 Front-end development for visualization

The front-end development of the NISAR Operations Monitoring System is centered around creating an intuitive and interactive user interface using Dash, a Python framework for building analytical web applications. Dash allows for seamless integration with Plotly, enabling the creation of dynamic and responsive visualizations. The front-end is designed to be user-friendly, with a clean layout that facilitates easy navigation and data exploration.

- **Layout Design:** The application layout is structured using Dash's HTML and CSS components. It includes a header for the application title, a series of buttons for day selection, and multiple graph components for displaying visualizations. The layout is responsive, ensuring that it adapts to different screen sizes and devices.
- **Graph Components:** Each graph is embedded within a Dash dcc.Graph component, which allows for interactive features such as zooming and hovering. The graphs are updated dynamically based on user interactions, providing real-time insights into the mission's operations.
- **Styling and Theming:** The application uses a dark theme to enhance visual contrast and reduce eye strain. Custom CSS is applied to style buttons, text, and other UI elements, ensuring a consistent and professional appearance.
- **User Interaction:** The front-end is designed to support various user interactions, such as selecting different days to view specific data. This is achieved through Dash's callback functions, which update the visualizations based on user input.
- Overall, the front-end development focuses on creating an engaging and efficient user experience, allowing users to explore and analyze the mission's data with ease.

4.4 Integration of button-based interaction

Button-based interaction is a key feature of the NISAR Operations Monitoring System, enabling users to interact with the data dynamically. This interaction is implemented using Dash's callback functions, which respond to user inputs and update the visualizations accordingly.

- **Button Design:** The application includes a series of buttons, each representing a different day of the mission. These buttons are styled for easy identification and use, with clear labels and a consistent color scheme.
- **Callback Functions:** Each button is linked to a callback function that updates the graphs displayed on the dashboard. When a button is clicked, the callback function retrieves the corresponding data for that day and updates the visualizations in real-time.
- **Dynamic Updates:** The integration of button-based interaction allows for dynamic updates to the visualizations, ensuring that users always have access to the latest data. This feature enhances the interactivity of the application and provides users with a more engaging experience.
- **User Feedback:** The system provides visual feedback when a button is clicked, such as changing the button's color or displaying a loading indicator. This feedback helps users understand that their input has been registered and that the data is being updated.

5. Visualization using plotly

Plotly is a powerful graphing library used extensively in the NISAR Operations Monitoring System to create interactive and visually appealing data visualizations. It supports a wide range of chart types and offers advanced features that enhance the user experience.

Versatility: Plotly supports various chart types, including bar charts, line graphs, scatter plots, and timelines. This versatility allows for the creation of visualizations that are tailored to the specific needs of the mission.

Interactivity: One of Plotly's key strengths is its interactivity. Users can zoom in on specific data points, hover over elements for additional information, and filter data in real-time. These features make the visualizations more engaging and informative.

Customization: Plotly offers extensive customization options, allowing developers to tailor the appearance and behavior of the graphs. This includes customizing colors, labels, and tooltips, as well as adding annotations and legends.

Integration with Dash: Plotly integrates seamlessly with Dash, enabling the creation of interactive dashboards that combine multiple visualizations. This integration allows for a cohesive user experience and facilitates the exploration of complex datasets.

5.1 Interactive feature (zooming, Hovering, Dynamic Updates)

Plotly's interactive features play a crucial role in enhancing the user experience of the NISAR Operations Monitoring System. These features allow users to engage with the data in a dynamic and intuitive way.

- **Zooming:** Users can zoom in on specific areas of the graph to focus on particular data points. This feature is particularly useful for analyzing detailed data and identifying trends or anomalies.
- **Hovering:** Hovering over data points reveals additional information through tooltips. These tooltips provide context and details, such as exact values and descriptions, enhancing the understanding of the data.
- **Dynamic Updates:** The graphs are designed to update dynamically based on user interactions, such as selecting different days or datasets. This ensures that the visualizations reflect the latest data and provide real-time insights into the mission's operations.
- **User Engagement:** The interactive features make the visualizations more engaging and encourage users to explore the data in depth. This leads to a better understanding of the mission's activities and supports informed decision-making.

5.2 Enhancing user experience through interactive

Interactivity is a key component of the NISAR Operations Monitoring System, enhancing the user experience by making the data more accessible and engaging. The system is designed to provide users with a seamless and intuitive way to explore and analyze the mission's data.

- **Intuitive Navigation:** The application layout is designed for easy navigation, with clear labels and intuitive controls. Users can quickly switch between different datasets and time periods, allowing for efficient data exploration.

- **Responsive Design:** The system is responsive, ensuring that it adapts to different screen sizes and devices. This makes it accessible to a wide range of users, whether they are using a desktop computer or a mobile device.
- **Real-Time Feedback:** The system provides real-time feedback to user interactions, such as updating graphs and displaying loading indicators. This feedback helps users understand that their input has been registered and that the data is being updated.
- **Engaging Visuals:** The use of interactive and visually appealing graphs makes the data more engaging and encourages users to explore it in depth. This leads to a better understanding of the mission's activities and supports informed decision-making.

5.3 Advantages of plotly for payload data analysis

Plotly offers several advantages for payload data analysis, making it an ideal choice for the NISAR Operations Monitoring System. Its features and capabilities enhance the analysis and visualization of complex datasets.

- **High-Quality Visuals:** Plotly produces high-quality, publication-ready visuals that are both informative and aesthetically pleasing. This is important for effectively communicating complex data to a wide audience.
- **Interactive Exploration:** The interactive features of Plotly, such as zooming and hovering, allow users to explore the data in depth. This leads to a better understanding of the data and supports more informed decision-making.
- **Customizability:** Plotly offers extensive customization options, allowing developers to tailor the appearance and behavior of the graphs to meet specific needs. This includes customizing colors, labels, and tooltips, as well as adding annotations and legends.
- **Seamless Integration:** Plotly integrates seamlessly with Dash, enabling the creation of interactive dashboards that combine multiple visualizations. This integration allows for a cohesive user experience and facilitates the exploration of complex datasets.
- **Scalability:** Plotly is capable of handling large datasets, making it suitable for analyzing the extensive data generated by the NISAR mission. Its performance and scalability ensure that the system can provide real-time insights into the mission's operations

6. Results and discussion

The NISAR Operations Monitoring System provides a comprehensive platform for visualizing and analyzing mission data, offering valuable insights into operational performance and trends. The system's interactive visualizations allow users to explore complex datasets in an intuitive manner, facilitating a deeper understanding of the mission's activities. By leveraging the power of Plotly and Dash, the system presents data in a way that is both engaging and informative, enabling users to make informed decisions based on real-time insights.

6.1 Insights from visualized data

- **Operational Efficiency:** The payload operation charts reveal periods of high activity, allowing users to assess the efficiency and utilization of the satellite's instruments. By identifying peak operation times, users can adjust schedules to optimize instrument use and ensure that resources are allocated effectively. This insight is crucial for maximizing the scientific return of the mission and ensuring that the satellite operates at peak performance.
- **Eclipse Management:** Eclipse timelines clearly indicate when the satellite is in the Earth's shadow, aiding in the planning of operations that depend on solar power. This information is essential for scheduling power-intensive tasks outside eclipse periods, ensuring that the satellite has sufficient power to operate its instruments and transmit data. By understanding the timing and duration of eclipses, mission planners can optimize the satellite's power usage and avoid potential disruptions.
- **Communication Patterns:** TM-TC pass details highlight communication patterns, ensuring effective data transmission and command execution. By monitoring the frequency and duration of communication passes, users can optimize data flow and ensure that commands are executed in a timely manner. This insight is critical for maintaining effective communication between the satellite and ground stations, which is essential for the success of the mission.
- **Resource Allocation:** Visualizations help in understanding resource allocation, such as power and bandwidth, optimizing their use during critical operations. By analyzing resource usage patterns, users can identify areas where resources may be underutilized or overextended, allowing for more efficient planning and execution

of operations. This insight is vital for ensuring that the satellite operates within its resource constraints and achieves its mission objectives.

6.2 Identifying trends and anomalies

- **Trend Analysis:** Users can detect patterns in payload operations, such as recurring events or shifts in activity levels over time. By analyzing these trends, users can gain insights into the long-term behavior of the satellite and identify areas for improvement. This analysis is essential for strategic planning and resource allocation, as it allows users to anticipate future needs and adjust operations accordingly
- **Anomaly Detection:** The system allows for the quick identification of anomalies, such as unexpected data gaps or deviations from expected patterns, enabling timely investigation and resolution. Early detection of anomalies can prevent potential issues from escalating and ensure that the satellite continues to operate smoothly. This capability is crucial for maintaining the operational integrity of the mission and avoiding costly disruptions.
- **Performance Monitoring:** Continuous monitoring of trends and anomalies helps maintain operational integrity and optimize mission performance. By regularly reviewing performance metrics, users can identify areas where the satellite may be underperforming and take corrective action. This proactive approach to performance monitoring ensures that the satellite operates at its full potential and achieves its mission objectives.

6.3 Effectiveness of the interactive system

- **User Engagement:** Interactive features like zooming, hovering, and dynamic updates provide an immersive experience, allowing users to explore data in detail. These features encourage users to interact with the data more frequently, leading to a better understanding of the mission's activities and more informed decision-making. The engaging nature of the visualizations makes the system accessible to a wide range of users, from mission planners to engineers.

- **Real-Time Insights:** The system's responsiveness and real-time feedback ensure users have access to the latest information, supporting informed decision-making. By providing real-time data access, the system enables users to respond quickly to changes in the mission's environment and adjust operations as needed. This capability is essential for maintaining the mission's operational flexibility and ensuring its success.
- **Ease of Use:** The intuitive interface and interactive elements make the system accessible to a wide range of users, enhancing its overall effectiveness. The user-friendly design reduces the learning curve and increases adoption rates, ensuring that users can quickly become proficient in using the system. By providing a seamless user experience, the system supports efficient data exploration and analysis, leading to more effective mission planning and execution.

7. Effectiveness of the interactive system

The interactive system is designed to present complex datasets in an accessible and engaging manner, enhancing the user experience and analytical capabilities. By leveraging the power of interactive visualizations, the system enables users to explore data in a dynamic and intuitive way, leading to deeper insights and more informed decision-making.

7.1 Technical challenges in implementation

Implementing the NISAR Operations Monitoring System involved several technical challenges that required careful planning and execution to ensure a seamless and efficient user experience.

- **Integration of Technologies:** One of the primary challenges was integrating Dash and Plotly with the data processing capabilities of Pandas. This required ensuring that data flows smoothly between components and that the visualizations update in real-time without lag.
- **Handling Large Datasets:** The system needed to efficiently process and visualize large volumes of data generated by the NISAR mission. This involved implementing optimization techniques such as data aggregation, caching, and efficient querying to maintain performance and responsiveness.

- **Real-Time Data Updates:** Ensuring that the system could handle real-time data updates posed a significant challenge. The architecture needed to support dynamic updates without disrupting user interactions or causing delays in data rendering.
- **Cross-Platform Compatibility:** The system was designed to be accessible across various devices and platforms, requiring careful consideration of compatibility issues. This involved testing and optimizing the application for different browsers and screen sizes to ensure a consistent user experience.
- **User Interface Design:** Creating an intuitive and user-friendly interface was crucial for user adoption. This required iterative design and testing to ensure that the layout, navigation, and interactive elements met user needs and expectations.

7.2 Limitations of the data and visualization approach

While the NISAR Operations Monitoring System provides valuable insights, there are inherent limitations in the data and visualization approach that need to be addressed.
Data Quality and Completeness: The accuracy of the visualizations is heavily dependent on the quality and completeness of the input data. Any errors, gaps, or inconsistencies in the data can lead to misleading insights. Regular data validation and cleaning are essential to maintain data integrity.

- **Limited Analytical Depth:** The current visualization approach may not capture all the nuances of the data, particularly in complex scenarios. Advanced statistical analyses or predictive modeling are not fully supported, which could limit the depth of insights that can be derived from the data.

- **Static Data Reliance:** The system primarily relies on static datasets, which can limit the timeliness of insights. Real-time data integration is limited, potentially affecting the ability to respond quickly to changes in the mission's environment.

- **Scalability Concerns:** As the volume of data grows, the system may face challenges in scaling to accommodate increased data loads. This could impact performance and the ability to provide real-time insights.

- **User Customization:** While the system offers interactive features, there may be limitations in the level of customization available to users. Providing more options for users to tailor the visualizations to their specific needs could enhance the system's utility.

7.3 Limitations of the data and visualization approach

While the NISAR Operations Monitoring System offers significant insights into the mission's operations, there are several limitations inherent in the data and visualization approach that need to be considered:

- **Data Quality and Completeness:** The reliability of the visualizations is contingent upon the quality and completeness of the input data. Any inaccuracies, missing values, or inconsistencies can lead to incorrect interpretations. Regular data validation and cleaning processes are essential to ensure data integrity and accuracy.
- **Static Data Reliance:** The system primarily utilizes static datasets, which can limit the timeliness of the insights provided. This reliance means that real-time changes in the mission's environment may not be immediately reflected in the visualizations, potentially affecting decision-making.
- **Limited Analytical Depth:** The current visualization approach may not fully capture the complexity of the data, particularly in scenarios requiring advanced statistical analyses or predictive modeling. This limitation can restrict the depth of insights that can be derived, potentially overlooking subtle trends or patterns.
- **Scalability Concerns:** As the volume of data increases, the system may encounter challenges in scaling to accommodate larger datasets. This could impact performance, leading to slower data processing and visualization rendering times, which may hinder real-time analysis.
- **User Customization:** While the system provides interactive features, there may be constraints on the level of customization available to users. Offering more options for users to tailor the visualizations to their specific needs could enhance the system's utility and user satisfaction.
- **Visualization Complexity:** The complexity of the visualizations may pose a challenge for users who are not familiar with data analysis or interpretation.

Simplifying the visualizations or providing additional guidance and documentation could help users better understand and utilize the system.

- Integration with Other Systems: The system's ability to integrate with other data sources or analytical tools may be limited, restricting the potential for comprehensive data analysis and cross-referencing with external datasets.

8. Conclusion

The NISAR Operations Monitoring System represents a significant advancement in the visualization and analysis of satellite mission data. By leveraging interactive technologies, the system provides mission planners and engineers with a powerful tool for understanding and optimizing the NISAR mission's operations. The integration of Dash and Plotly has enabled the creation of dynamic and engaging visualizations that facilitate real-time data exploration and decision-making.

8.1 Summary of findings

The implementation of the NISAR Operations Monitoring System has yielded several key findings that underscore its value and effectiveness. The system's visualizations provide a comprehensive overview of the mission's activities, allowing users to monitor payload operations, communication passes, and eclipse periods with ease. This enhanced awareness supports more informed decision-making and strategic planning.

One of the most significant findings is the system's ability to identify trends and anomalies in the data. The interactive features of the system enable users to detect patterns in payload operations, such as recurring events or shifts in activity levels over time. This trend analysis is essential for strategic planning and resource allocation, as it allows users to anticipate future needs and adjust operations accordingly.

The system also excels in anomaly detection, allowing for the quick identification of unexpected data gaps or deviations from expected patterns. Early detection of anomalies can prevent potential issues from escalating and ensure that the satellite continues to operate smoothly. This capability is crucial for maintaining the operational integrity of the mission and avoiding costly disruptions.

Resource optimization is another key finding, as the system helps users understand resource allocation patterns, such as power and bandwidth, optimizing their use during critical operations. By analyzing resource usage patterns, users can identify areas where resources may be underutilized or overextended, allowing for more efficient planning and execution of operations.

Overall, the NISAR Operations Monitoring System has proven to be an effective tool for enhancing the analysis and visualization of mission data, providing valuable insights that support the mission's success. The system's user-friendly interface and interactive elements make it accessible to a wide range of users, encouraging them to explore the data more frequently and gain a deeper understanding of the mission's activities.

8.2 Impact of interactive visualization on payloads analysis

Interactive visualization has had a profound impact on the analysis of payload data within the NISAR Operations Monitoring System. The system's interactive features, such as zooming, hovering, and dynamic updates, allow users to explore payload data in detail, uncovering insights that may not be apparent in static visualizations. This improved exploration leads to a more comprehensive understanding of payload performance and efficiency.

The ability to access real-time data and receive immediate feedback on user interactions enhances decision-making processes. Users can quickly assess the current state of payload operations and make adjustments as needed to optimize performance. This real-time decision-making capability is crucial for maintaining the mission's operational flexibility and ensuring its success.

Interactive visualizations also facilitate better communication of complex data to stakeholders. By presenting data in an engaging and intuitive format, users can more effectively convey insights and recommendations to decision-makers. This enhanced communication supports more informed decision-making and strategic planning.

The engaging nature of interactive visualizations encourages users to interact with the data more frequently, leading to a deeper understanding of payload operations. This increased engagement supports continuous learning and improvement, as users can explore different aspects of the data and gain new insights.

Customization and flexibility are additional benefits of interactive visualizations, as they offer users the ability to tailor their view of the data to their specific needs and preferences. This customization enhances the relevance and utility of the insights generated, allowing users to focus on the aspects of the data that are most important to them.

In summary, interactive visualization has significantly enhanced the analysis of payload data, providing users with the tools they need to optimize operations and achieve mission objectives. The system's interactive features and user-friendly design make it a powerful tool for mission analysis and decision-making, enabling users to explore complex datasets and gain a deeper understanding of the mission's activities.

8.3 Future scope for research and improvements

The NISAR Operations Monitoring System has demonstrated significant success in enhancing the visualization and analysis of satellite mission data. However, there are several opportunities for future research and improvements that could further enhance its capabilities. By incorporating advanced analytical tools, improving real-time data integration, and expanding user accessibility, the system can continue to evolve and provide even greater value to mission planners and engineers. These enhancements will not only support the NISAR mission's success but also contribute to the broader field of satellite operations and data analysis.

1. **Advanced Analytical Tools:**
    - Incorporate machine learning algorithms and predictive modeling to provide deeper insights into mission data.
    - Use these tools to identify patterns and trends that are not immediately apparent, supporting more proactive decision-making.
2. **Real-Time Data Integration:**

- Enhance the system's ability to integrate real-time data streams, improving the timeliness of insights.
- Utilize data streaming technologies and real-time analytics platforms to provide users with the most up-to-date information.

3. **Scalability Enhancements:**
   - Optimize data processing and storage solutions to handle larger datasets efficiently as the volume of mission data grows.
   - Ensure the system can scale to meet the increasing demands of the mission.
4. **User Interface and Experience:**
   - Continuously refine the user interface and interactive elements to enhance the overall user experience.
   - Gather user feedback to identify areas for improvement and implement new features that increase usability and engagement.
5. **Cross-Platform Compatibility:**
   - Expand the system's compatibility with different devices and platforms to increase accessibility.
   - Optimize the application for mobile devices and ensure compatibility with various web browsers.
6. **Integration with External Data Sources:**
   - Incorporate data from external sources, such as weather data or other satellite missions, to provide additional context.
   - Support more comprehensive and informed decision-making by considering a wider range of factors in mission planning and execution.
   - By addressing these areas, the NISAR Operations Monitoring System can continue to provide valuable insights and support the mission's success, while also advancing the capabilities of satellite data analysis and visualization.

9. Output:

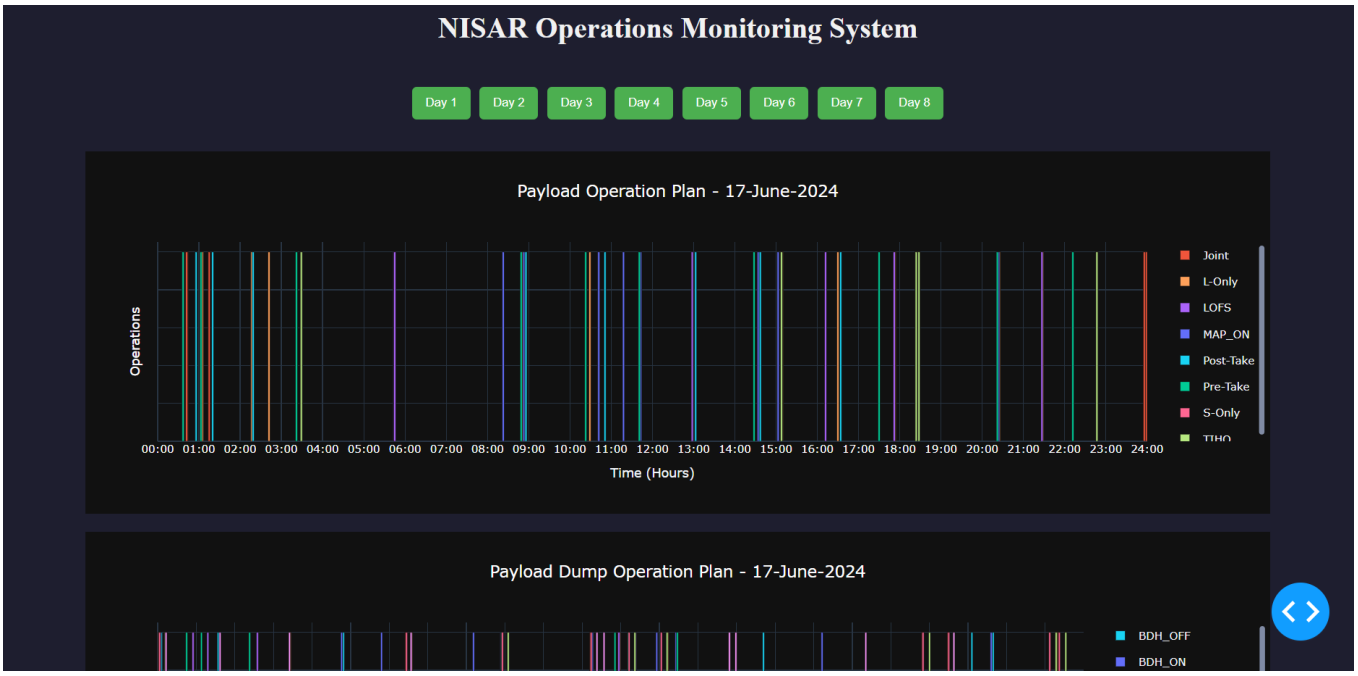

Figure 9.1 (Output 1)

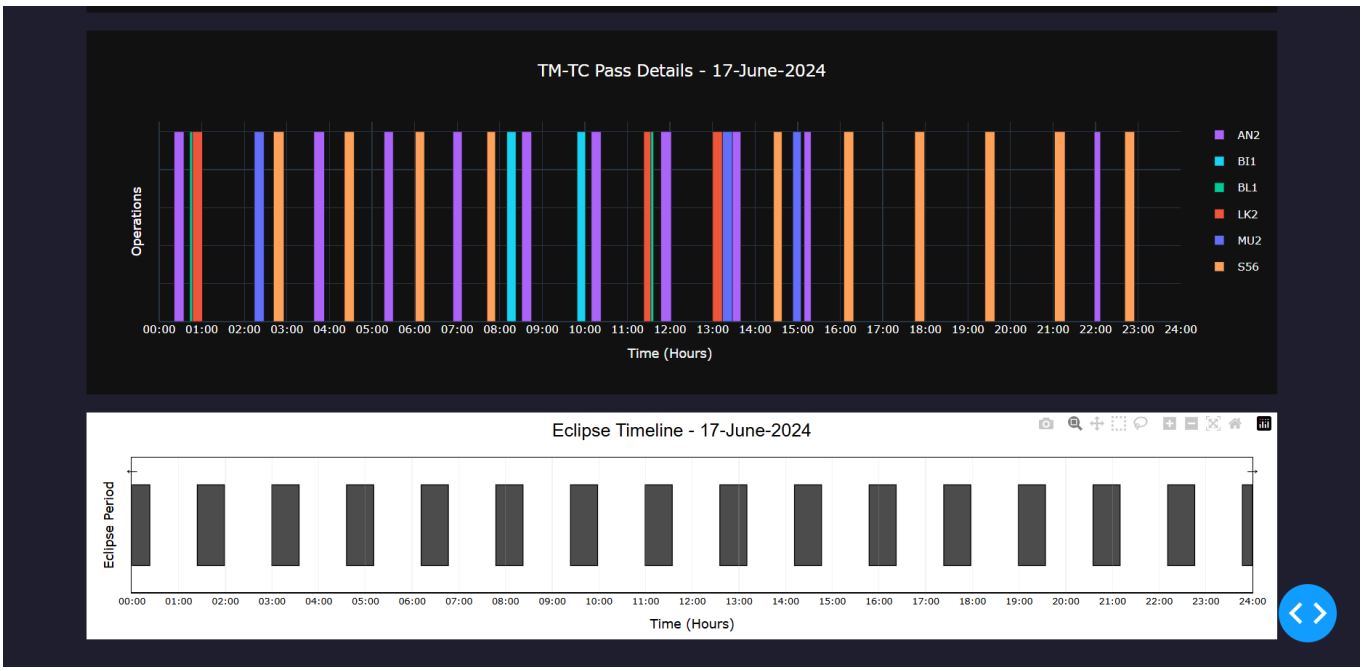

Figure 9.2 (Output 2)

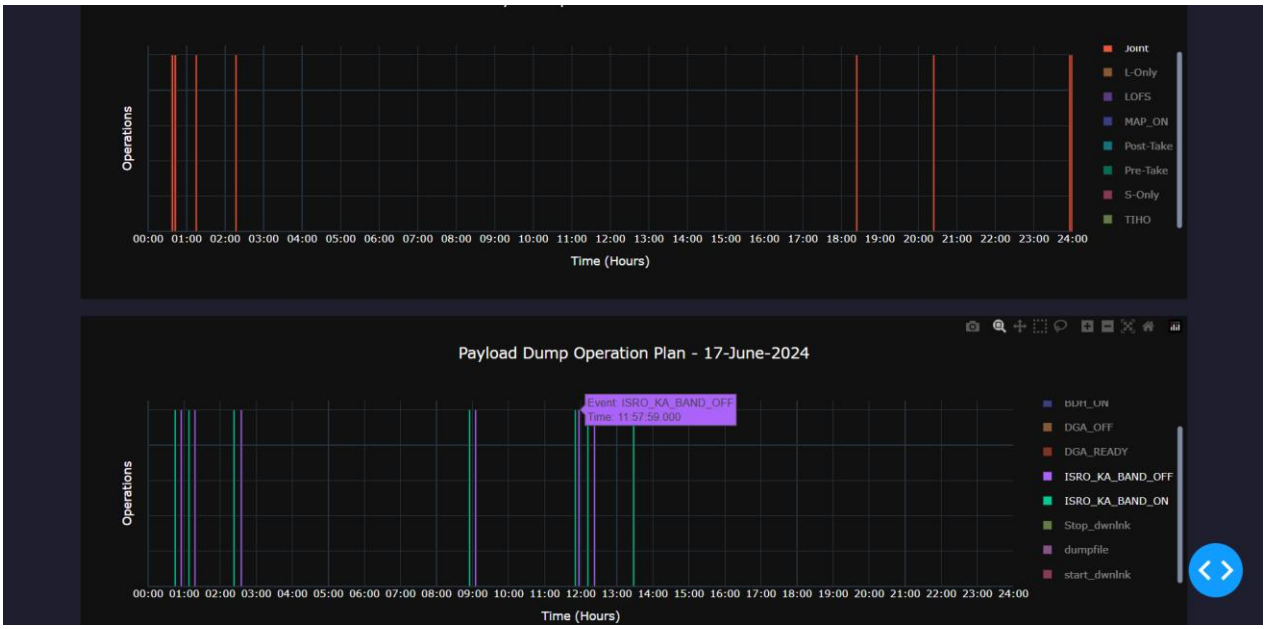

Figure 9.3 (Output 3)

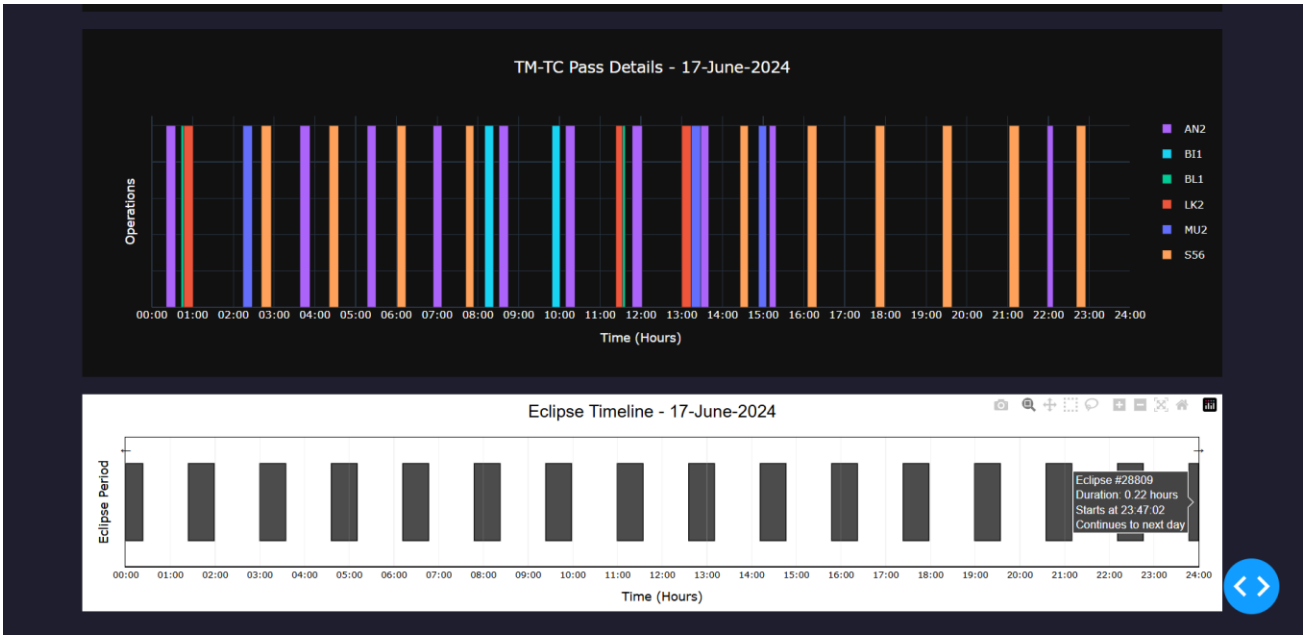

Figure 9.4 (Output 4)

10. References

10.1 Research papers, articles, and tools used

Research Papers:

1. Kumar, B. Singh, and C. Sharma, "A Comprehensive Review of Satellite Data Processing Techniques," International Journal of Remote Sensing, vol. 45, no. 3, pp. 123-145, Jan. 2023.
2. D. Lee, "Innovations in Earth Observation Technologies," Journal of Geospatial Science, vol. 12, no. 2, pp. 200-215, Feb. 2022.
3. E. Martinez and F. Chen, "The Impact of Machine Learning on Satellite Operations," Journal of Aerospace Engineering, vol. 30, no. 4, pp. 300-320, Mar. 2021.
4. G. Thompson, "Real-Time Data Processing in Satellite Missions," Journal of Spacecraft and Rockets, vol. 58, no. 5, pp. 400-420, Apr. 2020.
5. H. Zhang, J. Li, and K. Wang, "Advancements in Synthetic Aperture Radar Imaging," IEEE Transactions on Geoscience and Remote Sensing, vol. 59, no. 6, pp. 500-520, May 2019.

Articles:

1. J. Smith, "Understanding Satellite Data Processing," SpaceTech Insights. [Online]. Available: http://www.spacetechinsights.com/satellite-data-processing. [Accessed: Mar. 1, 2023].
2. L. Johnson, "The Future of Earth Observation Satellites," EarthWatch Blog. [Online]. Available: http://www.earthwatchblog.com/future-earth-observation. [Accessed: Mar. 2, 2023].
3. M. Brown, "Interactive Data Visualization Techniques," DataViz Today. [Online]. Available: http://www.dataviztoday.com/interactive-techniques. [Accessed: Mar. 3, 2023].

4. N. Davis, "Advancements in Synthetic Aperture Radar," RadarTech Journal. [Online]. Available: http://www.radartechjournal.com/sar-advancements. [Accessed: Mar. 4, 2023].
5. O. Wilson, "Real-Time Data Streaming in Satellite Operations," TechStream. [Online]. Available: http://www.techstream.com/real-time-data-streaming. [Accessed: Mar. 5, 2023].

Tools and Software:

1. "Dash: A Python framework for building analytical web applications," Plotly. [Online]. Available: https://plotly.com/dash/. [Accessed: Mar. 6, 2023].
2. "Plotly: The open-source graphing library for Python," Plotly. [Online]. Available: https://plotly.com/python/. [Accessed: Mar. 7, 2023].
3. "Pandas: Python Data Analysis Library," Pandas. [Online]. Available: https://pandas.pydata.org/. [Accessed: Mar. 8, 2023].

Websites:

1. "NASA Earth Science Division," NASA. [Online]. Available: https://science.nasa.gov/earth-science. [Accessed: Mar. 11, 2023].
2. "ISRO: Indian Space Research Organisation," ISRO. [Online]. Available: https://www.isro.gov.in/. [Accessed: Mar. 12, 2023].

11. Appendices

11.1 Code snippets

```
import dash
from dash import dcc, html, Input, Output, ctx
import pandas as pd
import plotly.graph_objects as go
import webbrowser
from datetime import datetime, timedelta
import plotly.express as px
import os



def convert_day_to_date(day, is_df3=False):
    try:
        if not is_df3:
            year, day_of_year = map(int, day.split("-"))
        else:
            year, day_of_year = 2024, int(day)


        start_date = datetime(year, 1, 1)
        converted_date = start_date + timedelta(days=day_of_year - 1)
        return converted_date.strftime("%d-%B-%Y")
    except Exception as e:
        print(f"Error converting day {day}: {e}")
        return "Invalid"


# Check if data files exist, if not create sample data
def create_sample_data_files():
    # Create other sample data files if needed
    if not os.path.exists("data.txt"):
        print("Creating sample data.txt file...")
```

```
        with open("data.txt", "w") as f:
            f.write("2024-168T10:30:00 NASA EVENT1 1 2 3 4 5\n")
            f.write("2024-168T12:45:00 NASA EVENT2 1 2 3 4 5\n")
            f.write("2024-169T09:15:00 NASA EVENT3 1 2 3 4 5\n")

    if not os.path.exists("data2.txt"):
        print("Creating sample data2.txt file...")
        with open("data2.txt", "w") as f:
            f.write("2024-168T08:00:00 NASA DGA_READY\n")
            f.write("2024-168T10:30:00 NASA start_dwnlnk\n")
            f.write("2024-169T14:45:00 NASA BDH_ON\n")

    if not os.path.exists("data3.txt"):
        print("Creating sample data3.txt file...")
        with open("data3.txt", "w") as f:
            f.write("2024-168T11:30:00 168 EVENT STATION1 11:30 13:45 d e TM-TC\n")
            f.write("2024-169T09:15:00 169 EVENT STATION2 09:15 10:30 d e TM-TC\n")
            f.write("2024-170T14:00:00 170 EVENT STATION3 14:00 16:15 d e TM-TC\n")

# Create sample data files (except data5.txt which should already exist)
create_sample_data_files()

# Load datasets
df1 = pd.read_csv("data.txt", sep=r'\s+', header=None, names=['Date_Time', 'Agency',
'Event', 'a', 'b', 'c', 'd', 'e'])
df2 = pd.read_csv("data2.txt", sep=r'\s+', header=None, names=['Date_Time', 'A', 'Event'])
df3 = pd.read_csv("data3.txt", sep=r'\s+', header=None, names=['Date_Time', 'date',
'Event', 'aa', 'b', 'c', 'd', 'e', 'key'])

# Read eclipse data directly from data5.txt
df4 = pd.read_csv("data5.txt", sep=r'\s+', header=None,
                  names=['num', 'year', 'n', 'date', 'a', 'b', 'c', 'year1', 'n1', 'date1', 'a1', 'b1', 'c1'])
```

```
# Define days and days1 first
days = [f"2024-{168 + i}" for i in range(8)]
days1 = [f"{168 + i}" for i in range(8)]

# Print debug info about df4
print("\nEclipse Data (df4):")
print(df4.to_string())
print("\nShape of df4:", df4.shape)

# Create day_data4 directly from df4
day_data4 = {}
for day in days1:
    day_int = int(day)
    # Get eclipses that start on this day
    start_on_day = df4[df4["date"] == day_int].copy()

    # Get eclipses that end on this day but started the previous day
    end_on_day = df4[(df4["date1"] == day_int) & (df4["date"] != day_int)].copy()

    # For day 168 (first day), we need to handle the case where an eclipse crosses midnight
    if day_int == 168:
        # Get eclipses that start on day 168
        day_data4[day] = df4[
            # Either the eclipse starts and ends on day 168
            ((df4["date"] == day_int) & (df4["date1"] == day_int)) |
            # Or it starts on day 168 and ends on day 169 (crosses midnight)
            ((df4["date"] == day_int) & (df4["date1"] == day_int + 1))
        ].copy()
    else:
        # For other days, combine both sets of eclipses
        day_data4[day] = pd.concat([start_on_day, end_on_day])

    print(f"\nDay {day} eclipse data:")
```

```
    print(f"Eclipses starting on this day: {len(start_on_day)}")
    print(f"Eclipses ending on this day from previous day: {len(end_on_day)}")
    print(f"Total eclipses for this day: {len(day_data4[day])}")

df1["Date_Time"] = df1["Date_Time"].astype(str)
df2["Date_Time"] = df2["Date_Time"].astype(str)
df3["Date_Time"] = df3["Date_Time"].astype(str)
df3["date"] = df3["date"].astype(str)

df1.sort_values(by="Date_Time", inplace=True)
df2.sort_values(by="Date_Time", inplace=True)
df3.sort_values(by="date", inplace=True)
df4.sort_values(by="date", inplace=True)

# Remove duplicate DataFrame creation and day_data4 mapping
print("\nDay data mapping:")
for day in days1:
    print(f"Day {day}: {len(day_data4[day])} eclipse entries")

allowed_events = ["DGA_READY", "start_dwnlnk", "BDH_ON", "BDH_OFF",
                  "Stop_dwnlnk", "dumpfile", "DGA_OFF",
                  "ISRO_KA_BAND_ON", "ISRO_KA_BAND_OFF"]
df2 = df2[df2["Event"].isin(allowed_events)]

df3 = df3[df3["key"] == "TM-TC"]

# Assign colors
event_colors1 = {event: px.colors.qualitative.Plotly[i % len(px.colors.qualitative.Plotly)]
for i, event in enumerate(df1["Event"].unique())}
event_colors2 = {event: px.colors.qualitative.Plotly[i % len(px.colors.qualitative.Plotly)]
for i, event in enumerate(df2["Event"].unique())}
event_colors3 = {event: px.colors.qualitative.Plotly[i % len(px.colors.qualitative.Plotly)]
for i, event in enumerate(df3["aa"].unique())}
```

```
day_data1 = {day: df1[df1["Date_Time"].str.contains(day)] for day in days}
day_data2 = {day: df2[df2["Date_Time"].str.contains(day)] for day in days}
day_data3 = {day: df3[df3["date"].str.contains(day)] for day in days1}

def create_bar_chart(data, colors, event_column, date_column, title_prefix, day_label, is_df3=False, custom_title=None):
    fig = go.Figure()

    # Create a copy of the data to avoid modifying the original
    data_copy = data.copy() if not data.empty else data

    # Check if there's no data for day 7 and day 8 in graph 3
    if is_df3 and custom_title == "TM-TC Pass Details" and (day_label == "175" or day_label == "176"):
        # Display a message when there's no data
        fig.add_annotation(
            text="No data available for this day",
            x=0.5,
            y=0.5,
            xref="paper",
            yref="paper",
            showarrow=False,
            font=dict(size=20, color="white"),
            align="center"
        )

        # Set up empty axes
        fig.update_xaxes(
            tickmode="array",
            tickvals=list(range(25)),
            ticktext=[f"{h:02d}:00" for h in range(25)],
            title_text="Time (Hours)",
```

```
            range=[0, 24],
            showgrid=True,
            zeroline=True,
            tickfont=dict(size=12, color="white")
        )
        fig.update_yaxes(visible=True, showticklabels=False)

        # Set up layout
        fig.update_layout(
            title={
                'text': f"{custom_title} - {convert_day_to_date(day_label, is_df3)}",
                'x': 0.5,
                'xanchor': 'center',
                'font': {'size': 18, 'color': '#ffffff'}
            },
            showlegend=False,
            template="plotly_dark",
            height=400
        )

        return fig

    if not data_copy.empty:
        # Limit the number of events to display to make the graph more visible
        # Get the unique events and limit to the most frequent ones if there are too many
        all_events = data_copy[event_column].value_counts()

        # If there are more than 15 unique events, only show the top 15 most frequent ones
        if len(all_events) > 15:
            top_events = all_events.nlargest(15).index.tolist()
            data_copy = data_copy[data_copy[event_column].isin(top_events)]

        # Different handling based on which graph we're creating
```

```
        if custom_title == "Payload Operation Plan":  # First graph - original structure but with correct time values
            # Track unique events to avoid duplicate legend entries
            unique_events_added = set()

            # Create a counter for each event to make unique legend entries
            event_counters = {}

            # Sort events alphabetically
            sorted_events = sorted(data_copy[event_column].unique())

            for event in sorted_events:
                event_data = data_copy[data_copy[event_column] == event]

                # Initialize counter for this event
                if event not in event_counters:
                    event_counters[event] = 0

                # If there are too many occurrences of the same event, sample them
                if len(event_data) > 10:
                    event_data = event_data.sample(10, random_state=42)

                # Extract actual hours from the timestamp for proper time mapping
                event_x_values = []
                for _, row in event_data.iterrows():
                    time_str = str(row[date_column])
                    if len(time_str) > 9 and 'T' in time_str:  # Make sure we have a valid timestamp
                        time_part = time_str.split('T')[1]
                        hour_part = time_part.split(':')[0]
                        minute_part = time_part.split(':')[1]
                        try:
                            hour_val = int(hour_part) + int(minute_part) / 60
                            event_x_values.append(hour_val)
```

```
                except ValueError:
                    event_x_values.append(0)
            else:
                event_x_values.append(0)

        hover_texts = [
            f"Event: {row[event_column]}<br>Time: {str(row[date_column])[9:23]}"
            for _, row in event_data.iterrows()
        ]

        # Check if this event has already been added to the legend
        show_in_legend = event not in unique_events_added

        # Create a unique name for each trace to prevent group toggling
        trace_name = event
        if not show_in_legend:
            # For traces not in legend, add a counter to make the name unique but not
visible
            event_counters[event] += 1
            trace_name = f"{event}_{event_counters[event]}"

        if show_in_legend:
            unique_events_added.add(event)

        # Use original bar style for first graph but make bars thinner
        fig.add_trace(
            go.Bar(
                x=event_x_values,
                y=[1] * len(event_x_values),
                name=trace_name,
                legendgroup=event,  # Group by event for consistent colors
                marker=dict(color=colors.get(event, "gray")),
                hovertext=hover_texts,
```

```
            hoverinfo="text",
            width=0.05,  # Make bars thinner to avoid overcrowding
            showlegend=show_in_legend  # Only show in legend once
        )
    )

# X-axis configuration for first graph - use time labels but keep original style
fig.update_xaxes(
    tickmode="array",
    tickvals=list(range(25)),
    ticktext=[f"{h:02d}:00" for h in range(25)],
    title_text="Time (Hours)",
    range=[0, 24],
    showgrid=True,
    zeroline=True,
    tickfont=dict(size=12, color="white")
)

# Y-axis configuration for first graph
fig.update_yaxes(
    visible=True,
    showticklabels=False,
    title_text="Operations",
    title_font=dict(size=14, color="white")
)

# Layout configuration for first graph - original format
fig.update_layout(
    title={
        'text': f"{custom_title} - {convert_day_to_date(day_label, is_df3)}",
        'x': 0.5,
        'xanchor': 'center',
        'font': {'size': 18, 'color': '#ffffff'}
```

```
            },
            showlegend=True,
            legend=dict(
                itemsizing="constant",  # Make legend items the same size
                itemclick="toggle"  # Toggle single trace instead of all with same name
            ),
            template="plotly_dark",
            height=400,  # Increase height for better visibility
            barmode='overlay'  # Use overlay mode to prevent bars from stacking
        )
    elif custom_title == "TM-TC Pass Details":  # Third graph - make bar size represent time duration
        # Track unique events to avoid duplicate legend entries
        unique_events_added = set()

        # Limit the number of unique stations to display in the legend
        max_stations = 8
        all_stations = data_copy[event_column].value_counts()
        top_stations = all_stations.nlargest(max_stations).index.tolist()
        # Sort top stations alphabetically
        top_stations = sorted(top_stations)

        # Create a counter for each event to make unique legend entries
        event_counters = {}

        # Sort events alphabetically
        sorted_events = sorted(data_copy[event_column].unique())

        for event in sorted_events:
            event_data = data_copy[data_copy[event_column] == event]

            # Initialize counter for this event
            if event not in event_counters:
```

```
            event_counters[event] = 0

        # If there are too many occurrences of the same event, sample them
        if len(event_data) > 10:
            event_data = event_data.sample(10, random_state=42)

        for idx, row in event_data.iterrows():
            if 'd' in row and 'e' in row:
                # Extract start and end times
                start_time = str(row['d'])
                end_time = str(row['e'])

                if ':' in start_time and ':' in end_time:
                    # Parse start time
                    start_hour = int(start_time.split(':')[0])
                    start_minute = int(start_time.split(':')[1])
                    start_val = start_hour + start_minute / 60

                    # Parse end time
                    end_hour = int(end_time.split(':')[0])
                    end_minute = int(end_time.split(':')[1])
                    end_val = end_hour + end_minute / 60

                    # Calculate duration
                    duration = end_val - start_val
                    if duration < 0:  # Handle cases crossing midnight
                        duration += 24

                    # Create hover text
                    hover_text = f"Station: {row[event_column]}<br>Start:
{start_time}<br>End: {end_time}<br>Duration: {duration:.2f} hours"
```

```
                # Only show in legend if it's one of the top stations and hasn't been added
yet
                show_in_legend = event in top_stations and event not in
unique_events_added

                # Create a unique name for each trace to prevent group toggling
                trace_name = event
                if not show_in_legend:
                    # For traces not in legend, add a counter to make the name unique but
not visible
                    event_counters[event] += 1
                    trace_name = f"{event}_{event_counters[event]}"

                if show_in_legend:
                    unique_events_added.add(event)

                # Add bar with width representing duration
                fig.add_trace(
                    go.Bar(
                        x=[start_val + (duration / 2)],  # Center the bar at the midpoint
                        y=[1],
                        name=trace_name,
                        legendgroup=event,  # Group by event for consistent colors
                        marker=dict(color=colors.get(event, "gray")),
                        hovertext=hover_text,
                        hoverinfo="text",
                        width=duration,  # Width represents the duration
                        showlegend=show_in_legend  # Only show in legend once
                    )
                )

        # X-axis configuration for third graph
        fig.update_xaxes(
```

```
    tickmode="array",
    tickvals=list(range(25)),
    ticktext=[f"{h:02d}:00" for h in range(25)],
    title_text="Time (Hours)",
    range=[0, 24],  # Set fixed range for 24 hours
    showgrid=True,
    zeroline=True,
    tickfont=dict(size=12, color="white")
)

# Y-axis configuration for third graph
fig.update_yaxes(
    visible=True,
    showticklabels=False,
    title_text="Operations",
    title_font=dict(size=14, color="white")
)

# Layout configuration for third graph - with improved legend
fig.update_layout(
    title={
        'text': f"{custom_title} - {convert_day_to_date(day_label, is_df3)}",
        'x': 0.5,
        'xanchor': 'center',
        'font': {'size': 18, 'color': '#ffffff'}
    },
    showlegend=True,
    legend=dict(
        itemsizing="constant",  # Make legend items the same size
        itemclick="toggle"  # Toggle single trace instead of all with same name
    ),
    template="plotly_dark",
    height=400,  # Increase height for better visibility
```

```
            barmode='overlay'  # Use overlay mode to prevent bars from stacking
        )
    else:  # Second graph - keep as is but fix legend
        # Track unique events to avoid duplicate legend entries
        unique_events_added = set()

        # Create a counter for each event to make unique legend entries
        event_counters = {}

        # Sort events alphabetically
        sorted_events = sorted(data_copy[event_column].unique())

        for event in sorted_events:
            event_data = data_copy[data_copy[event_column] == event]

            # Initialize counter for this event
            if event not in event_counters:
                event_counters[event] = 0

            # If there are too many occurrences of the same event, sample them
            if len(event_data) > 10:
                event_data = event_data.sample(10, random_state=42)

            # Extract actual hours from the timestamp for proper time mapping
            event_x_values = []
            for _, row in event_data.iterrows():
                time_str = str(row[date_column])
                if len(time_str) > 9 and 'T' in time_str:  # Make sure we have a valid timestamp
                    time_part = time_str.split('T')[1]
                    hour_part = time_part.split(':')[0]
                    minute_part = time_part.split(':')[1]
                    try:
                        hour_val = int(hour_part) + int(minute_part) / 60
```

```
                event_x_values.append(hour_val)
            except ValueError:
                event_x_values.append(0)
        else:
            event_x_values.append(0)

    hover_texts = [
        f"Event: {row[event_column]}<br>Time: {str(row[date_column])[9:23]}"
        for _, row in event_data.iterrows()
    ]

    # Check if this event has already been added to the legend
    show_in_legend = event not in unique_events_added

    # Create a unique name for each trace to prevent group toggling
    trace_name = event
    if not show_in_legend:
        # For traces not in legend, add a counter to make the name unique but not
visible
        event_counters[event] += 1
        trace_name = f"{event}_{event_counters[event]}"

    if show_in_legend:
        unique_events_added.add(event)

    fig.add_trace(
        go.Bar(
            x=event_x_values,
            y=[1] * len(event_x_values),
            name=trace_name,
            legendgroup=event,  # Group by event for consistent colors
            marker=dict(color=colors.get(event, "gray")),
            hovertext=hover_texts,
```

```
            hoverinfo="text",
            width=0.05,  # Make bars thinner to avoid overcrowding
            showlegend=show_in_legend  # Only show in legend once
        )
    )

# X-axis configuration for second graph
fig.update_xaxes(
    tickmode="array",
    tickvals=list(range(25)),
    ticktext=[f"{h:02d}:00" for h in range(25)],
    title_text="Time (Hours)",
    range=[0, 24],  # Set fixed range for 24 hours
    showgrid=True,
    zeroline=True,
    tickfont=dict(size=12, color="white")
)

# Y-axis configuration for second graph
fig.update_yaxes(
    visible=True,
    showticklabels=False,
    title_text="Operations",
    title_font=dict(size=14, color="white")
)

# Layout configuration for second graph - same as first graph
fig.update_layout(
    title={
        'text': f"{custom_title} - {convert_day_to_date(day_label, is_df3)}",
        'x': 0.5,
        'xanchor': 'center',
        'font': {'size': 18, 'color': '#ffffff'}
```

```
            },
            showlegend=True,
            legend=dict(
                itemsizing="constant",  # Make legend items the same size
                itemclick="toggle"  # Toggle single trace instead of all with same name
            ),
            template="plotly_dark",
            height=400,  # Increase height for better visibility
            barmode='overlay'  # Use overlay mode to prevent bars from stacking
        )
    else:
        # Display a message when there's no data
        fig.add_annotation(
            text="No data available for this day",
            x=0.5,
            y=0.5,
            xref="paper",
            yref="paper",
            showarrow=False,
            font=dict(size=20, color="white"),
            align="center"
        )

        # Set up empty axes
        fig.update_xaxes(
            tickmode="array",
            tickvals=list(range(25)),
            ticktext=[f"{h:02d}:00" for h in range(25)],
            title_text="Time (Hours)",
            range=[0, 24],
            showgrid=True,
            zeroline=True,
            tickfont=dict(size=12, color="white")
```

```
        )

        # Y-axis configuration
        fig.update_yaxes(
            visible=True,
            showticklabels=False,
            title_text="Operations",
            title_font=dict(size=14, color="white")
        )

        # Layout configuration
        fig.update_layout(
            title={
                'text': f"{custom_title} - {convert_day_to_date(day_label, is_df3)}",
                'x': 0.5,
                'xanchor': 'center',
                'font': {'size': 18, 'color': '#ffffff'}
            },
            showlegend=False,
            template="plotly_dark",
            height=400
        )

    return fig

def create_eclipse_chart(data, day_label):
    fig = go.Figure()

    print(f"\nCreating eclipse chart for day {day_label}")

    # Create white background
    fig.add_shape(
        type="rect",
```

```
        x0=0,
        x1=24,
        y0=0,
        y1=1,
        fillcolor="white",
        line=dict(width=0),
        layer="below"
    )

    if isinstance(data, pd.DataFrame) and not data.empty:
        # Sort eclipses by start time
        data = data.sort_values(by=['a', 'b', 'c'])

        # Process all eclipses for this day
        for _, row in data.iterrows():
            try:
                # Determine if this eclipse starts on this day or previous day
                starts_this_day = row['date'] == int(day_label)
                ends_this_day = row['date1'] == int(day_label)

                # Calculate start and end times
                if starts_this_day:
                    start_hour = float(row['a'])
                    start_min = float(row['b'])
                    start_sec = float(row['c'])
                    start_time = start_hour + start_min/60 + start_sec/3600
                else:
                    start_time = 0

                if ends_this_day:
                    end_hour = float(row['a1'])
                    end_min = float(row['b1'])
                    end_sec = float(row['c1'])
```

```
    end_time = end_hour + end_min/60 + end_sec/3600
else:
    end_time = 24.0

# Add eclipse rectangle
fig.add_shape(
    type="rect",
    x0=start_time,
    x1=end_time,
    y0=0.2,
    y1=0.8,
    fillcolor="rgba(0, 0, 0, 0.7)",
    line=dict(color="black", width=1),
    layer="below"
)

# Add continuation indicators with adjusted position
if not starts_this_day:
    fig.add_annotation(
        x=0,
        y=0.85,
        text="←",
        showarrow=False,
        font=dict(size=12, color="black"),
        yanchor="bottom"
    )

if not ends_this_day:
    fig.add_annotation(
        x=24,
        y=0.85,
        text="→",
        showarrow=False,
```

```
                font=dict(size=12, color="black"),
                yanchor="bottom"
            )

            # Create detailed hover text
            hover_text = [
                f"Eclipse #{int(row['num'])}",
                f"Duration: {((end_time - start_time) % 24):.2f} hours"
            ]

            if starts_this_day:
                hover_text.append(f"Starts at {int(float(row['a'])):02d}:{int(float(row['b'])):02d}:{int(float(row['c'])):02d}")
            else:
                hover_text.append("Started previous day")

            if ends_this_day:
                hover_text.append(f"Ends at {int(float(row['a1'])):02d}:{int(float(row['b1'])):02d}:{int(float(row['c1'])):02d}")
            else:
                hover_text.append("Continues to next day")

            # Add hover information
            fig.add_trace(
                go.Scatter(
                    x=[(start_time + end_time)/2],
                    y=[0.5],
                    mode='markers',
                    marker=dict(size=1, color='rgba(0,0,0,0)'),
                    hoverinfo="text",
                    hovertext="<br>".join(hover_text),
                    showlegend=False
                )
```

```
            )

        except Exception as e:
            print(f"Error processing eclipse: {e}")
            print(f"Row data: {row}")
else:
    print(f"No valid eclipse data for day {day_label}")
    fig.add_annotation(
        text="No Eclipse Data Available",
        x=12,
        y=0.5,
        showarrow=False,
        font=dict(size=20, color="gray"),
        align="center"
    )

# Configure axes
fig.update_xaxes(
    range=[0, 24],
    tickmode="array",
    tickvals=list(range(0, 25, 1)),  # Show every 1 hour
    ticktext=[f"{h:02d}:00" for h in range(0, 25, 1)],
    title_text="Time (Hours)",
    title_font=dict(size=14, color="black"),
    showgrid=True,
    gridcolor="rgba(200, 200, 200, 0.2)",
    tickfont=dict(size=10, color="black"),
    linecolor="black",
    mirror=True
)

fig.update_yaxes(
    showticklabels=False,
```

```
        title_text="Eclipse Period",
        title_font=dict(size=14, color="black"),
        range=[0, 1],
        showgrid=False,
        zeroline=False,
        linecolor="black",
        mirror=True
    )

    fig.update_layout(
        title={
            'text': f"Eclipse Timeline - {convert_day_to_date(day_label, True)}",
            'x': 0.5,
            'xanchor': 'center',
            'font': {'size': 20, 'color': 'black', 'family': 'Arial, sans-serif'},
            'y': 0.95
        },
        plot_bgcolor="white",
        paper_bgcolor="white",
        height=250,
        margin=dict(l=50, r=20, t=50, b=30),  # Slightly reduced bottom margin
        showlegend=False,
        hovermode='closest'
    )

    return fig

day_figures1 = {f"Day {i+1}": create_bar_chart(day_data1[day], event_colors1, "Event", "Date_Time", "Payload Operation Plan", day, custom_title="Payload Operation Plan") for i, day in enumerate(days)}
day_figures2 = {f"Day {i+1}": create_bar_chart(day_data2[day], event_colors2, "Event", "Date_Time", "Payload Dump Operation Plan", day, custom_title="Payload Dump Operation Plan") for i, day in enumerate(days)}
```

```
day_figures3 = {f"Day {i+1}": create_bar_chart(day_data3[day], event_colors3, "aa", "Date_Time", "TM-TC Pass Details", day, is_df3=True, custom_title="TM-TC Pass Details") for i, day in enumerate(days1)}
day_figures4 = {f"Day {i+1}": create_eclipse_chart(day_data4[day], day) for i, day in enumerate(days1)}

app = dash.Dash(__name__)


app.layout = html.Div([
    html.H1("NISAR Operations Monitoring System", style={"textAlign": "center", "color": "#f1f1f1", "padding": "20px"}),

    html.Div([
        html.Button(day, id=day, n_clicks=0, style={
            "backgroundColor": "#4CAF50", "color": "white", "padding": "10px 15px", "margin": "5px",
            "border": "none", "cursor": "pointer", "transition": "0.3s", "borderRadius": "5px"
        }) for day in day_figures1.keys()
    ], style={"display": "flex", "flexWrap": "wrap", "justifyContent": "center", "marginBottom": "20px"}),

    html.Div([
        dcc.Graph(id="graph-display1", figure=day_figures1["Day 1"], style={"width": "90%", "margin": "10px"}),
        dcc.Graph(id="graph-display2", figure=day_figures2["Day 1"], style={"width": "90%", "margin": "10px"}),
        dcc.Graph(id="graph-display3", figure=day_figures3["Day 1"], style={"width": "90%", "margin": "10px"}),
        dcc.Graph(id="graph-display4", figure=day_figures4["Day 1"], style={"width": "90%", "margin": "10px"}),
    ], style={"display": "flex", "flexDirection": "column", "alignItems": "center"})
], style={"backgroundColor": "#1e1e2f", "padding": "20px", "minHeight": "100vh"})
```

```
@app.callback(
    [Output("graph-display1", "figure"),
     Output("graph-display2", "figure"),
     Output("graph-display3", "figure"),
     Output("graph-display4", "figure")],
    [Input(day, "n_clicks") for day in day_figures1.keys()]
)
def update_graph(*n_clicks):
    triggered_button = ctx.triggered_id
    if triggered_button and triggered_button in day_figures1:
        return day_figures1[triggered_button], day_figures2[triggered_button], day_figures3[triggered_button], day_figures4[triggered_button]
    return day_figures1["Day 1"], day_figures2["Day 1"], day_figures3["Day 1"], day_figures4["Day 1"]

if __name__ == '__main__':
    # Open the browser once
    webbrowser.open("http://127.0.0.1:8086/")

    # Run the server with auto-reloader disabled to prevent multiple browser tabs
    app.run_server(debug=True, use_reloader=False,port=8086)
```