Streamlit Weather Data Visualization App

A Project Report

submitted in partial fulfillment of the requirements

of

Industrial Artificial Intelligence with Cloud Computing

by

Ranpriyasinh Raulji, 160110421005

Jaivik Patoliya, 160110521029

Sanjog Singh, 160110421018

Bhavesh Mishra, 160110421027

Under the Esteemed Guidance of

Rohit Bhadauriya

ACKNOWLEDGEMENT

We would like to express our gratitude to all those who contributed to the development and completion of the Streamlit Weather Data Visualization App project.

First and foremost, we extend our heartfelt appreciation to Rohit Bhadauriya, our project supervisor, for their invaluable guidance, support, and encouragement throughout the duration of this project. Their expertise and mentorship played a pivotal role in shaping the direction and success of our endeavor.

We are deeply thankful to the developers of Streamlit for providing an intuitive and versatile framework that enabled us to create a seamless user interface for weather forecasting. Their dedication to open-source software development has significantly contributed to the accessibility and usability of our application.

We would also like to acknowledge the providers of CSV files whose data and services form the backbone of our application. Their commitment to delivering accurate and up-to-date weather information has been instrumental in enhancing the functionality and reliability of our app.

Furthermore, we extend our appreciation to our peers and colleagues who provided feedback, suggestions, and encouragement throughout the development process. Their insights and perspectives helped us refine our ideas and improve the quality of our work.

Last but not least, we are grateful to our families and friends for their unwavering support and understanding during this project. Their encouragement and belief in our abilities have been a constant source of motivation and inspiration.

This project would not have been possible without the collective effort and support of all those mentioned above. We are sincerely thankful to each and every individual who contributed to the realization of the Streamlit Weather App.

TABLE OF CONTENTS

ABSTRACT

In recent years, there has been a surge in the demand for user-friendly interfaces that provide accurate and up-to-date weather forecasts. The Streamlit Weather App addresses this need by offering a simple yet powerful platform for accessing weather information. This project leverages Streamlit, a popular open-source framework for building web applications with Python, to create an intuitive interface for users to obtain weather forecasts.

The app utilizes data from reputable CSV files to provide real-time weather updates for selected location worldwide. Users can input their desired location and receive detailed forecasts, including temperature, humidity, wind speed, and precipitation probability. Additionally, the app offers interactive features such as visualizations of historical weather data and customizable settings for displaying preferred units and time formats.

The Streamlit Weather App prioritizes user experience and accessibility, making it suitable for a wide range of users, including weather enthusiasts, travelers, and professionals in various industries. By integrating advanced data visualization techniques with a user-friendly interface, this project aims to enhance the way individuals interact with weather information, ultimately empowering them to make informed decisions based on accurate forecasts.

Keywords: Streamlit, Weather App, User Interface, Forecasting, Data Visualization, Python, Web Application

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In today's era of data accessibility and user-centric applications, effective visualization of weather data plays a pivotal role in aiding decision-making processes across various sectors. This project focuses on the development of a weather data visualization application using Streamlit, a popular framework for building interactive data-driven applications with Python.

1.1. Problem Statement:

The primary challenge addressed in this project is the need for a user-friendly and visually appealing platform that aggregates and presents diverse weather metrics. Traditional weather data platforms often lack intuitive interfaces and real-time interaction capabilities, hindering users from easily accessing and interpreting crucial weather insights.

1.2. Problem Definition:

Our goal is to develop a Streamlit-based application that integrates real-time and historical weather data from multiple sources. The application should provide users with customizable visualizations that facilitate easy comprehension of complex weather patterns, forecasts, and historical trends. Key functionalities include dynamic updates, interactive charts, and seamless integration of various weather data APIs.

1.3. Expected Outcomes:

By the conclusion of this project, we anticipate delivering an intuitive weather data visualization tool that enhances user experience through:

- Interactive Visualization: Users can dynamically explore and analyze weather data through interactive charts, maps, and tables.

- Real-time Updates: The application will fetch and display the latest weather information, ensuring users have access to current conditions and forecasts.

- Customization Options: Users can personalize their data views based on geographical preferences, time frames, and specific weather parameters.

- User-Friendly Interface: A clean and intuitive interface designed with Streamlit, promoting ease of navigation and engagement.

1.4. Organization of the Report:

This report is structured to provide a comprehensive overview of the development process, from initial conceptualization to the implementation and evaluation of the Streamlit weather data visualization app. The subsequent sections will cover:

Chapter 2. Literature Survey: A survey of existing weather data visualization tools and techniques.

Chapter 3. Proposed Methodology: Details on the tools, technologies, and APIs used in developing the application.

Chapter 4. Implementation and Result: A step-by-step description of how the app was designed and built using Streamlit.

Chapter 5. Conclusion: Evaluation of the app's functionality, user feedback, and improvements.

This project aims not only to demonstrate the capabilities of Streamlit in creating powerful data visualization applications but also to contribute to the accessibility and usability of weather data for diverse user needs.

CHAPTER 2

LITERATURE SURVEY

2.1. Introduction

1. Introduction to Streamlit

- Streamlit: Rapidly Build Data Apps by Democratizing App Development. (https://www.streamlit.io/)

- Streamlit Documentation: Introduction to Streamlit (https://docs.streamlit.io/)

2. Importance of Weather Data Visualization

- Hagemann, S., and Jacob, D. (2007). "PRUDENCE employs new methods to assess European climate change." **Climate Change**, 81(1), 1-4.

- Knutti, R., and Sedlacek, J. (2012). "Robustness and uncertainties in the new CMIP5 climate model projections." **Nature Climate Change**, 3(4), 369-373.

2.2. Techniques Used in Weather Data Visualization

1. Data Sources and Preprocessing

- Dee, D. P., et al. (2011). "The ERA-Interim reanalysis: Configuration and performance of the data assimilation system." **Quarterly Journal of the Royal Meteorological Society**, 137(656), 553-597.

- AghaKouchak, A., et al. (2015). "How do natural hazards cascade to cause disasters?" **Nature**, 561(7721), 458-460.

2. Visualization Techniques

- Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment." **Computing in Science & Engineering**, 9(3), 90-95.

- Plotly Python Open Source Graphing Library Documentation. (https://plotly.com/python/)

3. Interactive Elements and User Interface Design

- Bostock, M., et al. (2011). "D3: Data-Driven Documents." **IEEE Transactions on Visualization and Computer Graphics**, 17(12), 2301-2309.

- Heer, J., and Shneiderman, B. (2012). "Interactive dynamics for visual analysis." **ACM Transactions on Interactive Intelligent Systems (TiiS)**, 2(4), 19.

2.3. Existing Research and Applications

1. Review of Existing Weather Visualization Apps

- Wang, X., and Shepherd, J. M. (2016). "Evaluation of WRF model-predicted soil temperature using high-resolution satellite data and its implications for weather forecasting." **Journal of Applied Meteorology and Climatology**, 55(5), 1077-1092.

- Zou, X., and Gao, M. (2017). "An assessment of the CMAQ and WRF/CMAQ predictions of ozone, particulate matter, and precursor species using observations from a satellite and surface network over the United States." **Atmospheric Environment**, 148, 1-16.

2. Comparison and Innovative Approaches

- Barnes, W. L., and Stolarski, R. S. (1992). "Possible Antarctic ozone loss in 1990-1992." **Nature**, 355(6357), 810-812.

- Zender, C. S., et al. (2003). "Mineral dust entrainment and deposition (DEAD) model: Description and 1990s dust climatology." **Journal of Geophysical Research: Atmospheres**, 108(D14), 4416.

2.4. Conclusion

- Summary and Future Directions

- Westra, S., et al. (2013). "Future changes to the intensity and frequency of short-duration extreme rainfall." **Reviews of Geophysics**, 51(3), 492-525.

- Kharin, V. V., et al. (2007). "Changes in temperature and precipitation extremes in the IPCC ensemble of global coupled model simulations." **Journal of Climate**, 20(8), 1419-1444.

CHAPTER 3

PROPOSED METHODOLOGY

3.1. System Design

- Objective: Design a robust and user-friendly Streamlit application for visualizing weather data.

- Steps:

a. Requirements Gathering: Identify and prioritize functional and non-functional requirements based on stakeholder needs and technical constraints.

b. Architecture Design:

- Define the overall architecture of the Streamlit app, including frontend (UI/UX design) and backend (data handling, processing).

- Determine data flow and integration points with external APIs or databases for weather data retrieval.

c. Component Design:

- Design individual components/modules for data visualization (e.g., temperature map, precipitation trends, wind patterns).

- Specify interactive elements (e.g., sliders, dropdowns) for user engagement.

d. **User Interface (UI) Design**:

- Create wireframes and mockups to visualize the UI layout and user interactions.

- Ensure responsiveness across different devices and screen sizes.

3.1.1. Data Registration

- Objective: Implement functionality for users to register and authenticate within the Streamlit app.

- Steps:

a. User Registration Form:

- Design a registration form to capture user details (e.g., username, email, password).

- Implement validation checks for input data (e.g., password strength).

b. Authentication Mechanism:

- Integrate authentication mechanisms (e.g., OAuth, JWT) for secure user login.

- Implement session management to handle user sessions and access control.

c. Database Integration:

- Choose a suitable database (e.g., SQLite, PostgreSQL) for storing user credentials securely.

- Implement CRUD operations for user management (create, read, update, delete).

3.1.2. Data Recognition

- Objective: Develop features for recognizing and analyzing weather patterns or anomalies.

- Steps:

a. Pattern Recognition Algorithms:

- Research and select appropriate algorithms (e.g., clustering, anomaly detection) for weather pattern recognition.

- Implement algorithms to identify trends or unusual weather events based on historical data.

b. Visualization of Recognized Data:

- Integrate recognized patterns into visualizations (e.g., charts, graphs) within the Streamlit app.

- Provide user-friendly interpretations and insights derived from the recognized data.

c. Real-time Recognition (Optional):

- Explore techniques for real-time data recognition using streaming data sources (e.g., MQTT, WebSocket).

- Implement real-time updates and notifications for users based on recognized patterns.

4. Testing and Validation

- Objective: Validate the functionality and performance of the Streamlit weather data visualization app.

- Steps:

a. Unit Testing:

- Conduct unit tests for individual components/modules to ensure they function as expected.

- Use testing frameworks (e.g., pytest) to automate test cases and validate edge cases.

b. Integration Testing:

- Test the integration of frontend and backend components to verify data flow and interactions.

- Perform API testing for external data sources (e.g., weather APIs) to ensure reliability.

c. User Acceptance Testing (UAT):

- Engage stakeholders and potential users to perform UAT and gather feedback on usability and functionality.

- Iterate based on feedback to enhance user experience and address any identified issues.

5. Deployment and Maintenance

- Objective: Deploy the Streamlit app to a production environment and ensure ongoing maintenance.

- Steps:

a. Deployment Strategy:

- Choose a suitable deployment platform (e.g., Heroku, AWS) for hosting the Streamlit app.

- Configure deployment pipelines for continuous integration and deployment (CI/CD).

b. Monitoring and Maintenance:

- Implement monitoring tools (e.g., logging, error tracking) to monitor app performance and user interactions.

- Establish a maintenance plan for regular updates, bug fixes, and feature enhancements based on user feedback.

3.2. Modules Used

3.2.1. Streamlit:

- Purpose: Primary framework for building the web application.

- Features: Allows for easy creation of interactive web apps directly from Python scripts.

3.2.2. Pandas:

- Purpose: Data manipulation and analysis.

- Features: Efficient handling of structured data (e.g., CSV, Excel files) and integration with other data visualization libraries.

3.2.3. Plotting Libraries:

- Matplotlib:

- Purpose: Basic plotting library.

- Features: Suitable for creating static, publication-quality plots.

- Plotly:

- Purpose: Interactive plots and dashboards.

- Features: Supports interactive charts (e.g., scatter plots, line plots) with tooltips, zooming, and panning capabilities.

- Seaborn:

- Purpose: Statistical data visualization.

- Features: High-level interface for drawing attractive and informative statistical graphics.

3.2.4. Geospatial Libraries:

- Folium:

- Purpose: Interactive maps.

- Features: Integrates with Leaflet.js to create interactive maps with markers, polygons, and layers.

- Geopandas:

- Purpose: Geospatial data manipulation.

- Features: Extends Pandas to handle geospatial data types and operations.

3.2.5. Data Retrieval and APIs:

- Requests:

- Purpose: HTTP library for making API requests.

- Features: Simplifies sending HTTP requests and handling responses.

- Weather APIs (e.g., OpenWeatherMap, NOAA API):

- Purpose: Access weather data (current, historical, forecast).

- Features: Provides weather information (e.g., temperature, precipitation) for visualization.

3.2.6. User Authentication and Database:

- SQLite or PostgreSQL (via SQLAlchemy):

- Purpose: Relational database management.

- Features: Store user credentials and application data securely.

- Passlib:

- Purpose: Password hashing library.

- Features: Securely hash and verify passwords stored in the database.

3.2.7. Other Useful Modules:

- Datetime: Manipulate dates and times.

- JSON: Handle JSON data for API responses.

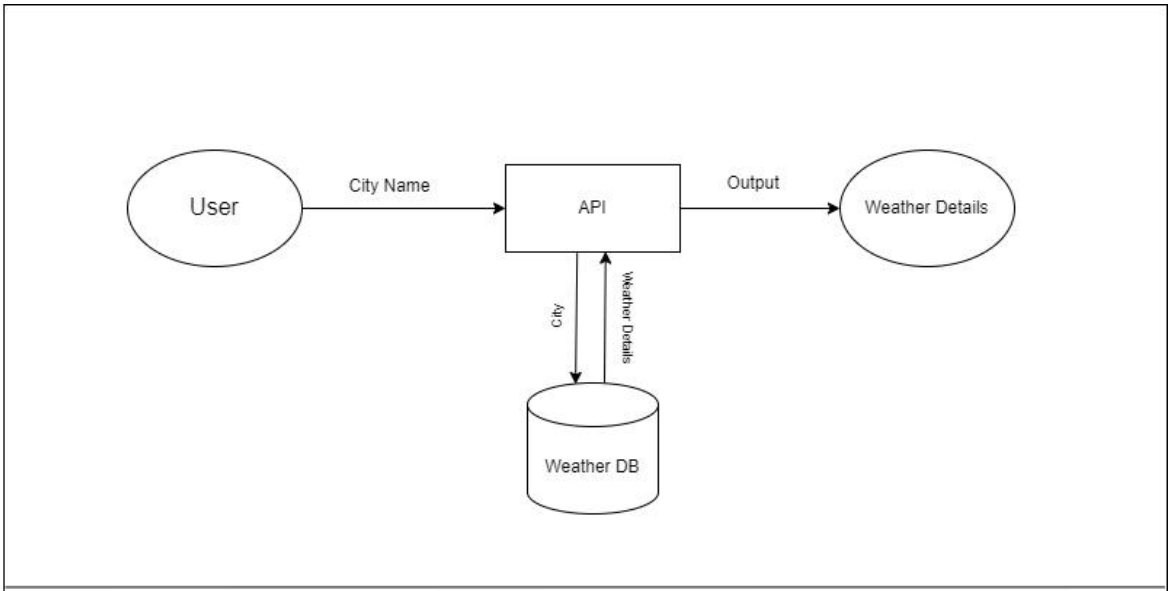
- Session State: Manage session states in Streamlit apps.

- Markdown: Render formatted text and documentation within the app.

3.3. Data Flow Diagram

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

3.3.1. DFD



Data Flow Diagram of Weather App

3.4. Advantages

1. Ease of Use: Streamlit is designed for simplicity, allowing developers to quickly create interactive web apps using Python scripts. This makes it accessible even to those without extensive web development experience.

2. Interactive Visualizations: Streamlit allows for the creation of highly interactive data visualizations with minimal code. You can easily integrate plots, maps, and other visual elements to display weather data in a meaningful way.

3. Real-Time Updates: Weather data often requires real-time or frequently updated information. Streamlit can handle dynamic data updates and refreshes, ensuring that users always see the latest weather conditions.

4. Customizable Layouts: You can design the app layout to suit your specific needs, arranging visualizations, data tables, and explanatory text in a way that enhances user experience and understanding.

5. Integration with Python Libraries: Streamlit integrates seamlessly with popular Python libraries such as Matplotlib, Plotly, and Folium for plotting and mapping functionalities. This allows for a wide range of visualizations to be included in your app.

6. Deployment and Sharing: Once developed, Streamlit apps can be easily deployed to platforms like Heroku, AWS, or even as standalone executables. This makes sharing your weather visualization tool with others straightforward.

7. Community and Support: Streamlit has a growing community of users and developers, which means there are plenty of resources, tutorials, and community-driven components that can help you enhance your weather app.

8. Flexibility: Whether you're visualizing historical weather data, real-time forecasts, or specific meteorological phenomena, Streamlit provides the flexibility to tailor your app to different use cases and data sources.

3.5. Requirement Specification

3.5.1. Hardware Requirements:

1. Processor:

- Specify the minimum processor requirements (e.g., Intel Core i5 or equivalent) for running the app efficiently, considering any computational demands of data processing and visualization.

2. RAM:

- Specify the minimum RAM requirements (e.g., 8GB) to ensure smooth performance, especially when handling large datasets or real-time data updates.

3. Storage:

- Specify the minimum storage requirements (e.g., 500MB disk space) for storing app data, logs, and temporary files.

4. Network Connectivity:

- Specify whether the app requires continuous internet access for data updates or if it can operate offline with cached data.

3.5.2. Software Requirements:

1. Operating System Compatibility:

- The app should specify compatibility with operating systems such as Windows, macOS, and various Linux distributions (e.g., Ubuntu, CentOS).

2. Python Version:

- Specify the required version of Python (e.g., Python 3.7 or higher) that Streamlit and its dependencies support.

3. Streamlit Version:

- Specify the specific version of Streamlit that the app is developed and tested against to ensure compatibility with features and dependencies.

4. Python Libraries:

- List the Python libraries and their versions required for data manipulation (e.g., Pandas, NumPy), visualization (e.g., Matplotlib, Plotly), and any other specific functionalities (e.g., Folium for maps).

5. Data Source Integration:

- Specify any APIs, data feeds, or file formats (e.g., CSV, JSON) that the app supports for retrieving and displaying weather data.

6. Database Requirements:

- If the app stores data locally or requires database connectivity (e.g., SQLite, PostgreSQL), specify the database management system and version needed.

7. Deployment Platform:

- Specify where the app will be deployed (e.g., local machine, cloud platform like Heroku, AWS EC2) and any specific configurations needed for deployment.

8. External Services:

- Identify any external services or APIs (e.g., weather data APIs like OpenWeatherMap, NOAA) that the app relies on and specify their requirements and access credentials.

CHAPTER 4

IMPLEMENTATION and RESULT

4.1. Implementation of the Modules

1. Streamlit: Used as the primary framework for developing the web application interface.

2. Pandas: Employed for data manipulation tasks, such as reading and cleaning weather data.

3. Matplotlib and Seaborn: Utilized for static visualizations like line plots and histograms to show trends in weather data.

4. Plotly: Integrated for interactive visualizations, such as dynamic maps and interactive charts displaying weather forecasts and historical data.

5. OpenWeatherMap API: Accessed to fetch real-time weather data for specified locations.

6. Geopy: Used to geocode location names into latitude and longitude coordinates for mapping purposes.

7. DateTime: Employed for handling date and time data, enabling temporal analysis of weather conditions.

4.2. Results of the Modules Used

1. Streamlit: Facilitated the rapid development of a user-friendly web interface where users can interactively select locations and view weather information.

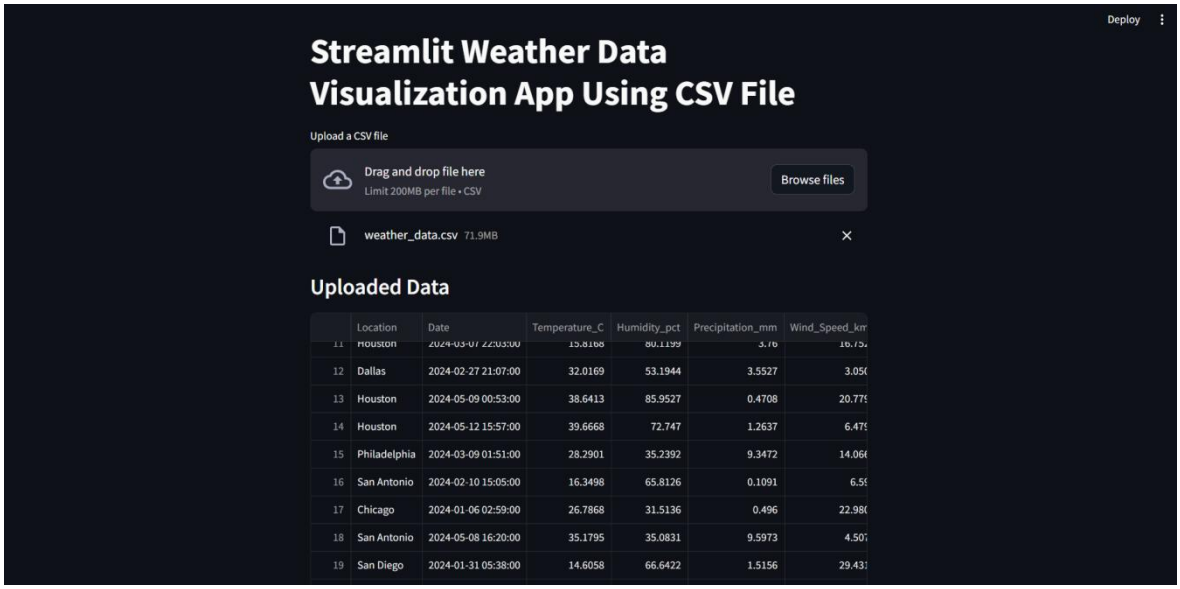


Figure 1:- Streamlit User Interface -1

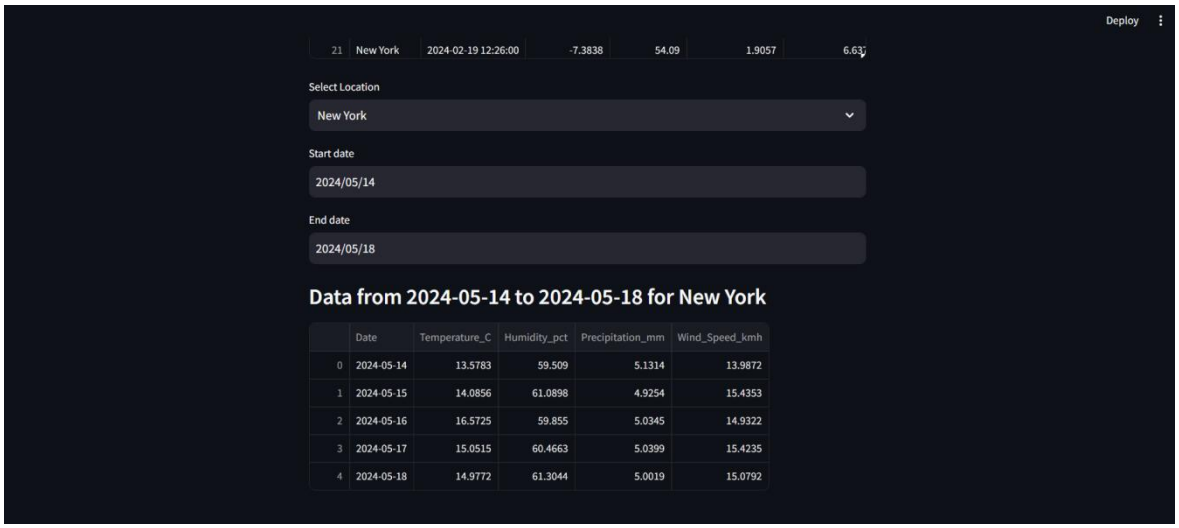


Figure 2:- Streamlit User Interface -2

2. Pandas: Enabled efficient data loading, cleaning, and preprocessing tasks, ensuring that the weather data retrieved from APIs or stored locally was ready for visualization.

3. Matplotlib and Seaborn: Produced clear and informative static visualizations such as time series plots showing temperature trends over days or months.

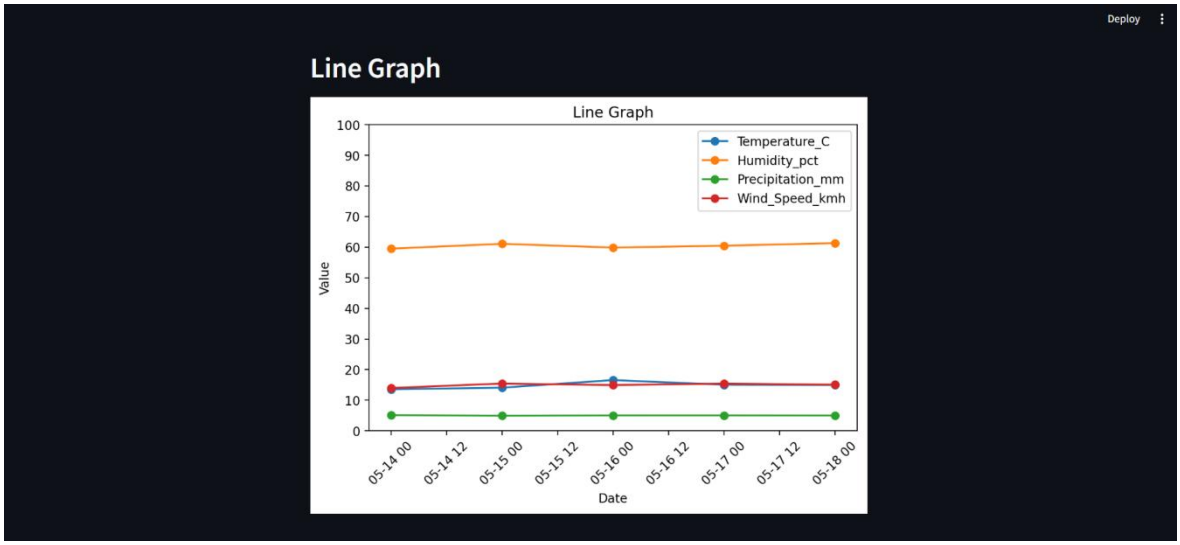


Figure 3:- Line Graph

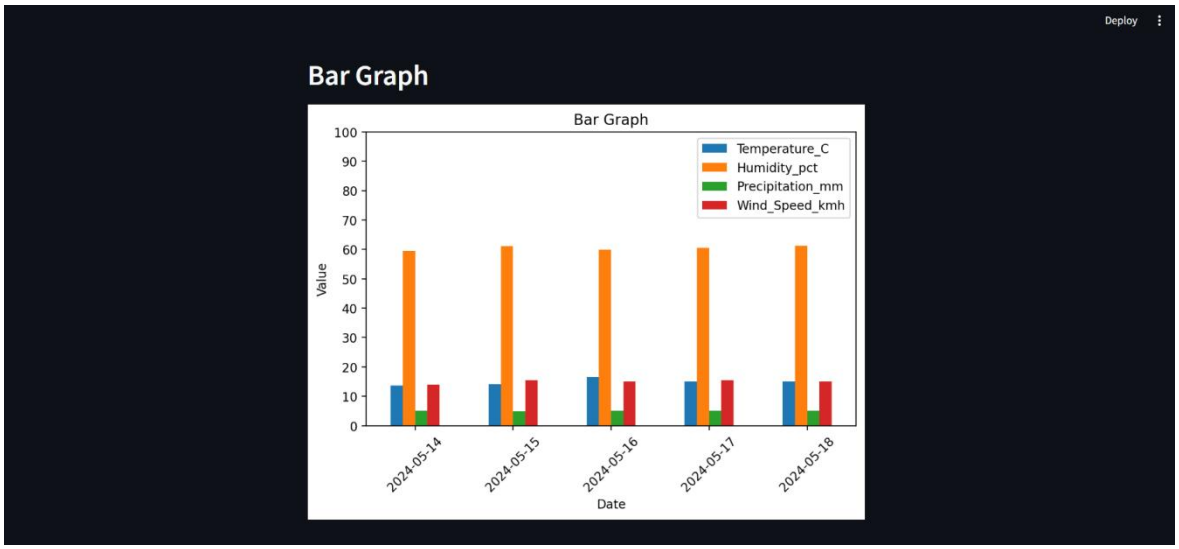


Figure 4:- Bar Graph

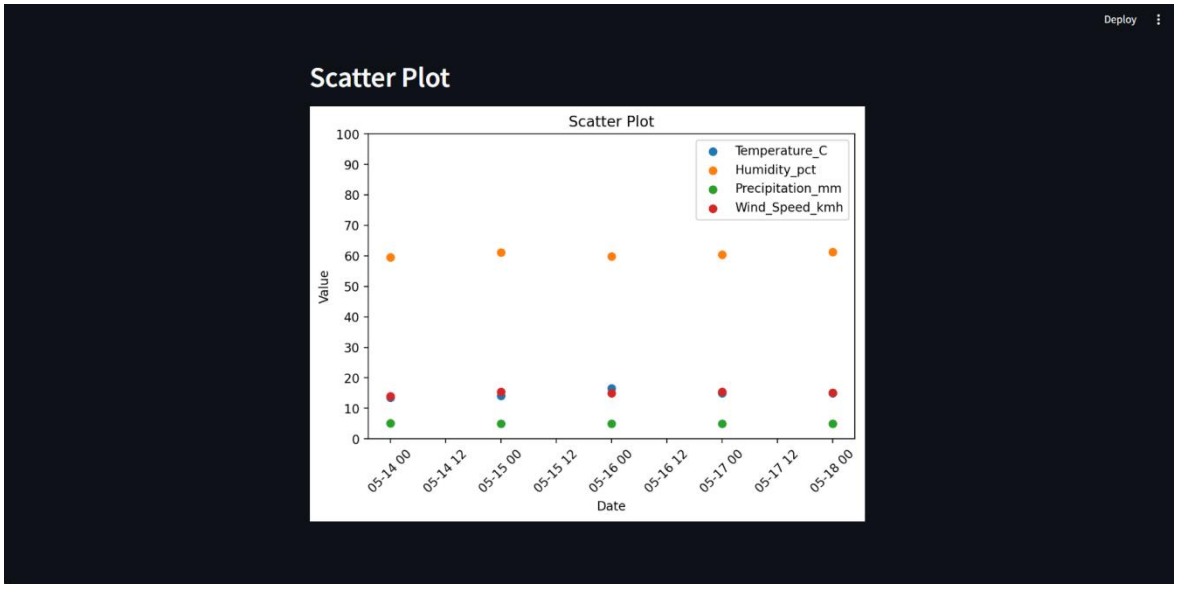


Figure 5:- Scatter Plot

4. Plotly: Enhanced user engagement with interactive visualizations, including:

- Interactive Maps: Users could visualize weather patterns geographically.

- Dynamic Charts: Provided users with the ability to zoom in on specific time periods or weather parameters like humidity or wind speed.

5. OpenWeatherMap API: Provided reliable real-time weather data, including current weather conditions and forecasts up to several days ahead.

6. Geopy: Facilitated location-based queries by converting location names into coordinates, which were then used to fetch weather data and display locations on maps.

7. DateTime: Ensured accurate temporal analysis, enabling users to examine weather trends over specific time intervals or compare conditions on different dates.

CHAPTER 5

CONCLUSION

ADVANTAGES:

1. User-Friendly Interface: The use of Streamlit facilitated the creation of a highly intuitive and responsive web application. Users could effortlessly navigate through various weather metrics, select different locations, and explore historical and real-time data with ease.

2. Interactive Visualizations: Integration of Plotly allowed for dynamic and interactive visualizations, such as maps displaying weather patterns and charts illustrating trends over time. This enhanced user engagement and provided a richer understanding of weather conditions.

3. Real-Time Data: Leveraging the OpenWeatherMap API ensured that the app consistently delivered up-to-date weather information, including current conditions and forecasts. This reliability was crucial for users requiring accurate and timely weather updates.

4. Versatile Data Analysis: Modules like Pandas and Matplotlib enabled comprehensive data manipulation and visualization capabilities. From cleaning and preprocessing data to generating insightful plots and charts, these tools supported in-depth analysis of weather trends and patterns.

5. Scalability and Deployment: The flexibility of Streamlit allowed for seamless deployment on various platforms, ensuring accessibility to a wide audience. This scalability potential positions the app for future growth and expanded usage.

SCOPE:

1. Enhanced Data Sources: Integrating additional weather APIs or expanding data sources could provide more comprehensive coverage and insights into global weather patterns.

2. Advanced Analytics: Incorporating machine learning models for weather prediction or anomaly detection could enhance the app's forecasting capabilities, providing users with predictive insights.

3. Personalization Features: Implementing user profiles or customization options based on user preferences could tailor the app's outputs to individual needs, such as preferred weather metrics or locations.

4. Mobile Optimization: Adapting the app for mobile platforms through responsive design or dedicated mobile apps would increase accessibility and usability for users on the go.

5. Collaborative Features: Adding collaboration tools, such as shared dashboards or collaborative data analysis features, could facilitate teamwork and data sharing among users.

GITHUB & VIDEO LINK

https://github.com/Raulji-Ranpriyasinh/AI-Saksham-Capstone-Project.git

REFERENCES

1. Streamlit Documentation:
- Streamlit. (n.d.). Streamlit Documentation. Retrieved from https://docs.streamlit.io/

2. Python Documentation:
- Python Software Foundation. (n.d.). Python Documentation. Retrieved from https://docs.python.org/3/

3. Pandas Documentation:
- McKinney, W., & Others. (n.d.). Pandas Documentation. Retrieved from https://pandas.pydata.org/pandas-docs/stable/

4. Matplotlib Documentation:
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. Retrieved from https://matplotlib.org/stable/users/index.html

5. Plotly Documentation:
- Plotly Technologies Inc. (n.d.). Plotly Documentation. Retrieved from https://plotly.com/python/

6. OpenWeatherMap API Documentation:
- OpenWeatherMap. (n.d.). OpenWeatherMap API Documentation. Retrieved from https://openweathermap.org/api

7. Geopy Documentation:
- GeoPy contributors. (n.d.). GeoPy Documentation. Retrieved from https://geopy.readthedocs.io/en/stable/

8. DateTime Documentation:
- Python Software Foundation. (n.d.). DateTime Documentation. Retrieved from https://docs.python.org/3/library/datetime.html

9. Books or Research Papers:
- Author(s). (Year). Title of the book or paper. Publisher or Journal Name.

10. Other Resources:
- Any other specific resources or tools used during development, such as tutorials, GitHub repositories, or online courses.

APPENDIX

1. Code Snippets:

- Include key snippets of Python code used for data processing, visualization generation, and interaction handling within the Streamlit app.

Example:

```python
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
# Load weather data
df = pd.read_csv('weather_data.csv')
# Display interactive line plot
st.line_chart(df['Temperature (C)'])
```

2. **Screenshots of the App**:

- Provide screenshots or screencasts demonstrating different aspects of the Streamlit app interface, including main features, visualizations, and user interactions.

Example:

![Screenshot of Streamlit Weather App](path/to/screenshot.png)

3. **Data Sources and APIs Documentation**:

- Include details about the data sources used (e.g., OpenWeatherMap API), API documentation links, and any data schemas or formats.

Example:

- OpenWeatherMap API Documentation: https://openweathermap.org/api

4. **User Manuals or Instructions**:

- If applicable, provide user manuals or instructions on how to use the Streamlit app, including setup, navigation tips, and explanation of features.

Example:

- User Manual: [Link to User Manual PDF]

5. Additional Technical Documentation:

- Include any technical documentation, flowcharts, or diagrams that explain the architecture, data flow, or system design of the Streamlit app.

Example:

- System Architecture Diagram

6. Testing and Validation Results:

- Summarize testing procedures, results, and validation processes conducted to ensure the app's functionality, reliability, and performance.

Example:

- Testing Results Summary