PROJECT REPORT

ON

WEATHER ANALYSIS

Submitted in the partial fulfilment of the requirement

for the award of degree of

BACHELOR OF TECHONOLOGY

IN

COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE

BATCH

(2022-2026)

SUBMITTED TO:

Ms.BINDU BANSAL

SUBMITTED BY:

SAGAR SIDHU

12200250

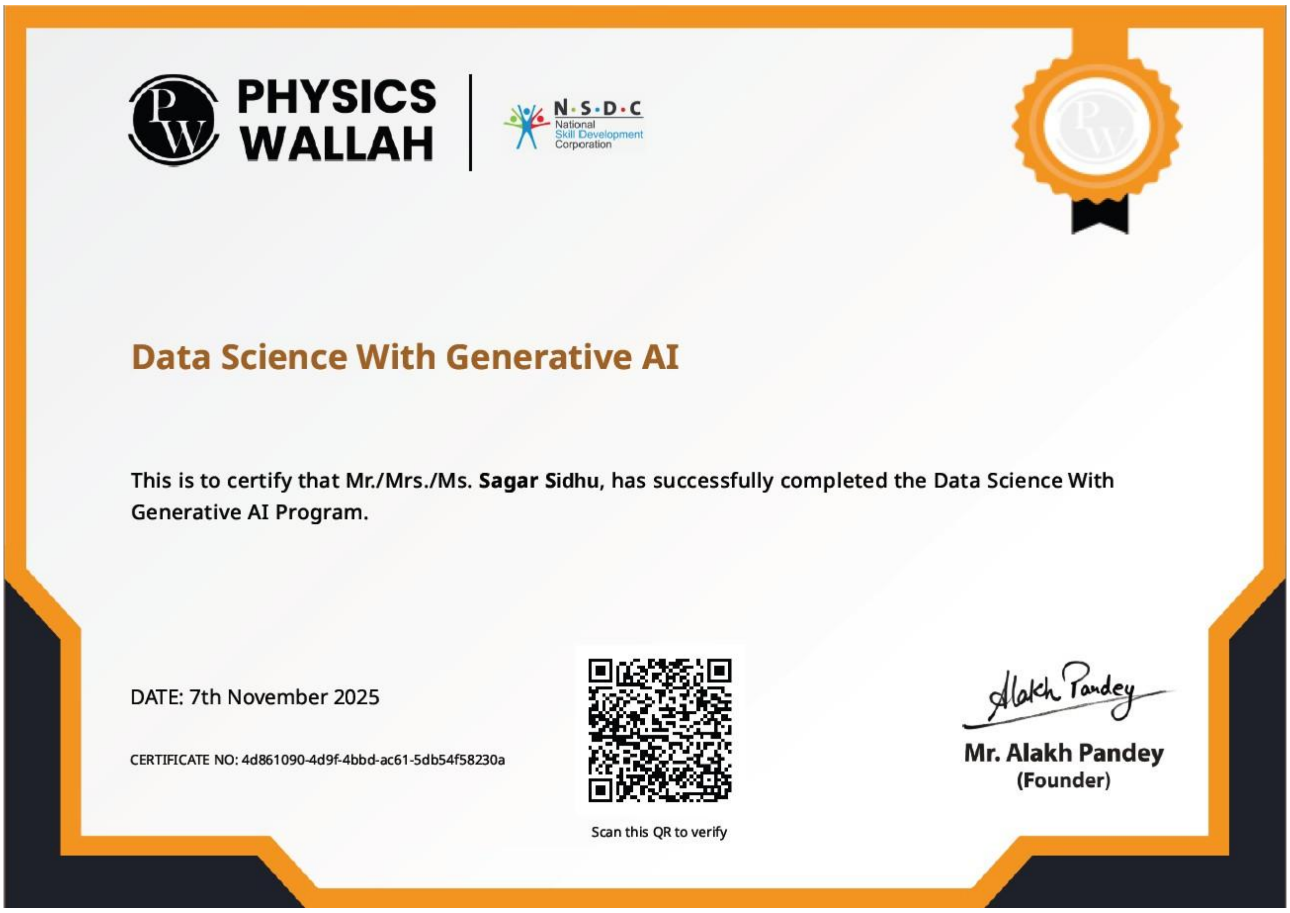
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DAV UNIVERSITY

JALANDHAR-PATHANKOT NATIONAL HIGHWAY, NH 44,

SARMASTPUR PUNJAB 144012

Training certificate



PHYSICS WALLAH

N·S·D·C National Skill Development Corporation

Data Science With Generative AI

This is to certify that Mr./Mrs./Ms. **Sagar Sidhu**, has successfully completed the Data Science With Generative AI Program.

DATE: 7th November 2025

CERTIFICATE NO: 4d861090-4d9f-4bbd-ac61-5db54f58230a

Scan this QR to verify

Alakh Pandey

Mr. Alakh Pandey
(Founder)

Table of content

Abstract

Weather analysis is a critical component in understanding and predicting climatic patterns and their impacts on various sectors such as agriculture, transportation, and public health. This project focuses on analysing weather data to identify trends and patterns, using a combination of data science techniques and tools. Leveraging Python along with libraries like pandas, numpy, matplotlib, seaborn, and Streamlit, we aim to perform comprehensive data analysis and visualization. Additionally, MySQL is utilized for efficient data storage and retrieval.

The analysis begins with data collection from reliable weather databases, followed by data cleaning and preprocessing to ensure accuracy and consistency. Key weather parameters such as temperature, humidity, precipitation, and visibility are examined over different time periods and geographic locations. The project employs statistical methods and visualization techniques to uncover trends, correlations, and anomalies in the weather data.

A significant part of this project involves analyzing visibility trends in Delhi, which has implications for air quality and transportation safety. By studying historical data, we can identify periods of low visibility and correlate them with factors such as pollution levels and seasonal changes. The results of this analysis are presented through an interactive Streamlit application, providing users with easy access to the findings and the ability to explore the data dynamically.

Overall, this project not only enhances our understanding of weather patterns but also provides actionable insights for decision-makers in various fields. The integration of advanced data analysis tools and techniques ensures that the analysis is thorough and reliable, contributing to the broader field of climatology and environmental science.

ACKNOWLEDGEMENT

Sagar Sidhu,The completion of "**Weather Analysis**" stands as a testament to the collective dedication and collaborative effort. I extend my heartfelt appreciation to all those who contributed significantly to this project.

My sincere gratitude goes to Mr. Ajay Kumar Gupta for his invaluable guidance, unwavering support, and profound insights throughout the development journey. His mentorship played a pivotal role in steering the project in the right direction and fostering an environment conducive to innovation and excellence. I express my appreciation to him for his unwavering commitment and collaborative spirit throughout the project's lifecycle. His perspective played a crucial role in shaping and refining the project's concepts and execution.

Special thanks go to my family and friends for their unwavering encouragement and understanding during the challenging phases of this Endeavour. Their support was a constant source of motivation.

Lastly, I am grateful to all those individuals who, directly or indirectly, contributed to this project. Your support has been instrumental, and I am truly thankful for your contributions.

Introduction

Weather analysis is a pivotal aspect of environmental science, playing a critical role in understanding atmospheric phenomena and their impacts on various sectors such as agriculture, transportation, health, and urban planning. As climate change and environmental concerns continue to rise, the importance of accurate and comprehensive weather analysis cannot be overstated. This project delves into the intricate patterns of weather data, utilizing advanced data science techniques to uncover trends and provide actionable insights.

The primary focus of this analysis is on key weather parameters such as temperature, humidity, precipitation, and visibility, which are crucial for predicting weather conditions and assessing environmental health. By leveraging Python and its powerful libraries like pandas, numpy, matplotlib, and seaborn, we conduct thorough data preprocessing, statistical analysis, and visualization. These tools enable us to handle large datasets efficiently and extract meaningful patterns that might otherwise remain obscured.

In addition, we employ MySQL for robust data management, ensuring that the data is stored securely and can be queried efficiently. Streamlit is used to create an interactive web application that allows users to explore the data and findings dynamically. This interactivity enhances the accessibility and usability of the analysis, making it valuable not only to researchers and policymakers but also to the general public.

A significant part of this project is dedicated to analyzing visibility trends in Delhi, a city grappling with severe air pollution issues. By examining historical weather data, we aim to identify periods of low visibility and their correlation with pollution levels and seasonal changes. This analysis provides crucial insights for improving air quality management and transportation safety.

Overall, this project underscores the transformative power of data science in weather analysis. By combining rigorous statistical methods with state-of-the-art visualization tools, we aim to contribute to the broader understanding of weather patterns and support efforts in mitigating the adverse effects of climate change.

4o

Objectives

The primary objectives of this weather analysis project are to comprehensively collect, integrate, and preprocess weather data from reliable sources, ensuring its accuracy and consistency. By focusing on key weather parameters such as temperature, humidity, precipitation, and visibility, we aim to identify significant trends and patterns across various time periods and geographic locations. A particular emphasis is placed on analyzing visibility trends in Delhi, correlating them with pollution levels and seasonal changes to provide actionable insights for air quality management and transportation safety. Utilizing MySQL for efficient data storage and management, we ensure robust data integrity and facilitate complex querying and analysis. Employing statistical methods and advanced visualization tools like matplotlib and seaborn, we seek to uncover correlations, anomalies, and significant trends within the weather data. The development of an interactive web application using Streamlit aims to make our findings dynamically accessible to researchers, policymakers, and the general public, enhancing the usability and impact of our analysis. Through this project, we strive to generate insights that inform decision-making in various sectors affected by weather conditions, such as agriculture, transportation, and public health, and to provide recommendations for mitigating the adverse effects of climate change. Ultimately, our goal is to contribute to the broader field of climatology and environmental science by sharing our methodologies and findings, supporting ongoing efforts in climate change mitigation and environmental health improvement.

Description

Weather analysis involves the systematic examination of atmospheric data to understand and predict weather patterns and their impacts on various aspects of life and the environment. This project aims to conduct a thorough analysis of weather data by leveraging advanced data science techniques and tools. The analysis is centered around key meteorological parameters such as temperature, humidity, precipitation, and visibility, which are critical for understanding weather conditions and their fluctuations.

Data Collection and Preprocessing: The process begins with collecting extensive weather data from reputable sources. This data is then cleaned and preprocessed to ensure it is accurate and consistent, removing any anomalies or errors that could skew the analysis. Preprocessing steps include handling missing values, normalizing data, and transforming it into a suitable format for analysis.

Exploratory Data Analysis (EDA): EDA is performed to gain an initial understanding of the data. This involves visualizing the distribution and relationships of different weather parameters using histograms, scatter plots, and correlation matrices. EDA helps in identifying patterns, trends, and anomalies in the data, setting the stage for more detailed analysis.

Trend and Pattern Analysis: Using statistical methods and time series analysis, the project aims to uncover long-term trends and seasonal patterns in weather data. For instance, trends in temperature and precipitation over the years can indicate changes in climate patterns, while seasonal variations can provide insights into the cyclic nature of weather phenomena.

Focus on Visibility Trends in Delhi: A significant aspect of this project is the detailed analysis of visibility trends in Delhi, a city known for its air pollution issues. By examining historical data on visibility, we aim to correlate low visibility periods with factors such as pollution levels and seasonal changes. This analysis can provide valuable insights for improving air quality management and transportation safety in the region.

Data Management and Storage: MySQL is utilized for efficient data storage and management. The structured storage of data facilitates easy querying and retrieval, enabling complex analyses and ensuring the integrity and security of the data.

Visualization and Interactive Reporting: Visualization plays a crucial role in making complex data understandable and actionable. Using Python libraries such as matplotlib and seaborn, we create comprehensive and clear visual representations of the data and analysis results. Additionally, an interactive web application is developed using Streamlit, allowing users to explore the data and findings dynamically. This enhances the accessibility of the analysis, making it valuable to a broader audience including researchers, policymakers, and the general public.

Insights and Recommendations: The insights derived from this analysis are intended to inform decisionmaking in various sectors impacted by weather conditions, such as agriculture, transportation, and public health. The project aims to provide recommendations for mitigating the adverse effects of climate change and improving environmental health.

Overall, this weather analysis project combines rigorous data science methodologies with advanced visualization techniques to provide a comprehensive understanding of weather patterns and trends. By focusing on actionable insights and interactive reporting, it aims to support efforts in climate change mitigation and environmental health improvement.

4o

Data Preprocessing

Data processing is a crucial step in weather analysis, involving the collection, cleaning, transformation, and analysis of weather data to extract meaningful insights. This process ensures that the data is accurate, consistent, and suitable for further analysis and visualization. Here's a detailed overview of the data processing steps involved in weather analysis:

1. Data Collection:

Sources: Weather data is collected from Kaggle

Types: Data typically includes parameters like temperature, humidity, precipitation, wind speed, and visibility. Historical weather data is also collected for trend analysis.

2. Data Integration:

- **Combining Datasets:** Multiple datasets from different sources or time periods are integrated into a unified format. This might involve merging data from weather stations, satellite imagery, and other sources.
- **Standardization:** Ensuring that all data follows a consistent format and unit of measurement. For example, temperatures might be standardized to Celsius or Fahrenheit.

3. Data Cleaning:

- **Handling Missing Values:** Missing or incomplete data is identified and addressed. Techniques include interpolation, where missing values are estimated based on surrounding data, or data imputation methods.
- **Removing Outliers:** Identifying and handling anomalous data points that deviate significantly from expected ranges. Outliers can be corrected, removed, or flagged for further investigation.
- **Error Correction:** Identifying and correcting errors in the data, such as incorrect timestamps or sensor malfunctions.

4. Data Transformation:

- **Normalization:** Scaling data to a standard range, especially if combining data from different sources with varying units or scales.
- **Aggregation:** Summarizing data over specific time intervals (e.g., daily, monthly) to simplify analysis. For example, hourly temperature readings might be aggregated to daily averages.
- **Feature Engineering:** Creating new variables or features from existing data to enhance analysis. For instance, calculating temperature anomalies or humidity indices.

5. Data Exploration:

- **Descriptive Statistics:** Calculating summary statistics such as mean, median, standard deviation, and range to understand the distribution of weather parameters.
- **Visualization:** Using plots and charts to visually inspect data distributions and relationships. Common visualizations include time series plots, histograms, scatter plots, and heatmaps.

6. Data Analysis:

- **Statistical Analysis:** Applying statistical methods to identify trends, correlations, and patterns. Techniques may include regression analysis, correlation analysis, and hypothesis testing.
- **Time Series Analysis:** Analyzing temporal patterns and trends in weather data over time. Methods include trend analysis, seasonal decomposition, and forecasting models.
- **Geospatial Analysis:** If applicable, analyzing weather data in the context of geographical locations using spatial analysis techniques.

7. Data Storage:

- **Database Management:** Storing processed data in a structured format using databases such as MySQL or PostgreSQL. This facilitates efficient querying, retrieval, and management of large datasets.

8. Data Visualization and Reporting:

- **Interactive Dashboards:** Creating interactive visualizations using tools like Streamlit, allowing users to explore and interact with the data dynamically.
- **Reports:** Generating detailed reports and summaries of findings, including visualizations and interpretations of the data analysis.

9. Quality Assurance:

- **Validation:** Ensuring the processed data meets quality standards and accurately represents the original data. This includes cross-verifying with external sources or benchmarks.
- **Documentation:** Documenting the data processing steps, methodologies, and any assumptions made during the analysis to ensure reproducibility and transparency.

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) in weather analysis involves examining and visualizing weather data to uncover underlying patterns, relationships, and anomalies. This process is crucial for understanding the dataset before performing more complex analyses and making informed decisions based on the findings. Here's an overview of how EDA is typically conducted in weather analysis:

1. **Data Overview:** Begin by reviewing the dataset's structure, including its size, types of variables, and basic statistics. This helps in understanding the breadth and depth of the data and identifying any immediate issues such as missing values or inconsistencies.
2. **Summary Statistics:** Calculate basic descriptive statistics for each weather parameter, such as mean, median, standard deviation, and range. This provides a general understanding of the distribution and central tendency of the data.
3. **Visualization:** Create visual representations of the data to identify patterns and relationships. Common visualizations in weather analysis include:
   - **Histograms:** To show the distribution of individual weather parameters like temperature or humidity.
   - **Box Plots:** To visualize the spread and identify outliers in parameters like precipitation or wind speed.
   - **Time Series Plots:** To examine how weather parameters change over time and to detect trends, seasonality, or periodicity.
   - **Scatter Plots:** To explore relationships between different weather variables, such as the correlation between temperature and humidity.
   - **Heatmaps:** To visualize correlations between multiple variables and identify patterns in large datasets.
4. **Seasonal Analysis:** Investigate how weather parameters vary with seasons or specific time periods. This can include analyzing monthly or quarterly trends to understand seasonal variations in temperature, precipitation, and other parameters.
5. **Anomaly Detection:** Identify and investigate any unusual or unexpected values in the data. This can involve spotting outliers in weather conditions or detecting extreme events like heatwaves or heavy rainfall.
6. **Correlation Analysis:** Assess the relationships between different weather variables using correlation coefficients. For instance, analyze how changes in humidity correlate with changes in precipitation or temperature.
7. **Data Aggregation:** Aggregate data at different time intervals (e.g., daily, weekly, monthly) to simplify analysis and uncover broader trends. This helps in smoothing out short-term fluctuations and focusing on long-term patterns.

8. **Data Transformation:** Apply transformations to make the data more suitable for analysis, such as normalizing values, creating new derived variables (e.g., dew point from temperature and humidity), or converting data to different units.
9. **Comparative Analysis:** Compare weather data across different regions or time periods to identify regional differences or temporal changes. This can reveal insights into how local weather patterns differ from global trends or how climate change impacts specific areas.
10. **Interaction with Data:** Use interactive tools and dashboards to explore the data dynamically. This allows users to filter, zoom, and drill down into specific aspects of the data to gain deeper insights and make data-driven decisions.

Through EDA, analysts gain a comprehensive understanding of the weather data, which informs subsequent analyses and helps in generating actionable insights for decision-making in various domains such as climate science, urban planning, and public health.

TECHNOLOGIES IN USE:

1. Python:

Python, created by Guido van Rossum in 1991, has evolved into one of the most popular programming languages worldwide. Its philosophy emphasizes code readability and simplicity, embodied in its clean syntax that often uses English keywords where other languages use punctuation. This design makes Python an excellent choice for beginners while still powerful enough for professional software development.

As an interpreted language, Python code is executed line by line, which facilitates rapid development and debugging. Its dynamic typing allows variables to change types, offering flexibility in coding. Python's object-oriented features support encapsulation, inheritance, and polymorphism, enabling developers to create complex, modular programs. However, it also supports other programming paradigms, including procedural and functional styles, giving developers the freedom to choose the most suitable approach for their tasks.

Python's standard library is often described as "batteries included" due to its comprehensive nature. It provides modules for file I/O, system calls, networking, and much more, reducing the need for external dependencies in many projects. Additionally, the Python Package Index (PyPI) hosts over 350,000 thirdparty packages, extending Python's capabilities to specialized domains like web development (Django, Flask), data analysis (NumPy, Pandas), machine learning (TensorFlow, PyTorch), and scientific computing (SciPy).

The language's cross-platform nature is a significant advantage, allowing code written on one operating system to run on others with little to no modification. This portability, combined with Python's simplicity, has led to its adoption in various fields, including web development, scientific research, artificial intelligence, and automation.

Python's memory management is handled by an automatic garbage collector, which frees developers from manual memory allocation and deallocation. This feature, along with built-in data structures like lists, tuples, sets, and dictionaries, simplifies complex data handling tasks.

The language's support for modules and packages allows for logical organization of code, promoting reusability and maintainability. This modular approach, coupled with Python's extensive documentation and active community, has created a rich ecosystem for developers.

Python's versatility is evident in its wide-ranging applications. In web development, frameworks like Django and Flask have become popular for building scalable web applications. In data science and machine learning, libraries such as NumPy, Pandas, and Scikit-learn have made Python the language of choice for many researchers and data analysts. The language is also widely used in scientific computing, game development, network programming, and system administration.

Recent versions of Python have introduced new features like f-strings for easier string formatting, the walrus operator for assignment expressions, and improved asynchronous programming support. These additions continue to enhance Python's capabilities while maintaining its core philosophy of simplicity and readability.

2. Streamit:

Streamlit is an open-source Python library that simplifies the creation of web applications for data science and machine learning projects. Launched in 2019, Streamlit has quickly gained popularity among data scientists and developers for its ability to turn data scripts into shareable web apps with minimal effort. At its core, Streamlit allows users to create interactive, data-driven web applications using pure Python code. This approach eliminates the need for front-end development skills typically required for web application development, such as HTML, CSS, or JavaScript. Instead, developers can focus on their data analysis and visualization code, while Streamlit handles the web interface generation.

Streamlit's architecture is designed for simplicity and rapid development. When a Streamlit script is run, it generates a web application where each variable or function creates a user interface element. The library provides a wide range of UI components, including text inputs, buttons, sliders, and file uploaders, which can be easily integrated into the app with just a few lines of code.

One of Streamlit's key features is its ability to automatically rerun the entire script when any input widget is changed. This reactive execution model ensures that the app's state is always consistent with user inputs, simplifying the development of interactive applications. Streamlit also supports caching to optimize performance for computationally intensive operations.

For data visualization, Streamlit seamlessly integrates with popular Python libraries such as Matplotlib, Plotly, and Altair. This allows developers to create complex, interactive charts and graphs within their Streamlit apps. The library also supports the display of multimedia content, including images, videos, and audio files.

Streamlit's ecosystem includes Streamlit Components, which allows developers to create custom widgets and extend the library's functionality. This feature enables the integration of more advanced UI elements or third-party visualizations into Streamlit apps.

Deployment of Streamlit apps is straightforward, with options for cloud deployment through services like Streamlit Cloud (formerly Streamlit Sharing) or self-hosting on platforms like Heroku or AWS. This ease of deployment facilitates sharing and collaboration, making it simple for data scientists to showcase their work to stakeholders or the wider community.

In the data science workflow, Streamlit serves as a powerful tool for creating prototypes, dashboards, and interactive reports. It's particularly useful for quickly iterating on machine learning models, allowing data scientists to easily create interfaces for model demonstration and evaluation.

Recent updates to Streamlit have introduced features like multipage apps, allowing for more complex application structures, and improvements in app performance and customization options. The library continues to evolve, with an active community contributing to its development and expanding its capabilities.

Streamlit's simplicity, coupled with its powerful features, has made it a popular choice in various industries, from finance and healthcare to education and research. It's particularly valued in scenarios where rapid prototyping and deployment of data-driven applications are crucial.

Streamlit

3. Pandas

Pandas is a powerful, open-source data manipulation and analysis library for Python. Created by Wes McKinney in 2008 and released publicly in 2009, pandas has become an essential tool in the data science ecosystem, bridging the gap between the scientific computing capabilities of NumPy and the flexible data manipulation requirements of data analysis.

At its core, pandas provides two primary data structures: Series (one-dimensional) and DataFrame (twodimensional). The DataFrame, in particular, has become ubiquitous in data analysis, offering a tabular, spreadsheet-like data structure with labeled axes (rows and columns). This structure allows for intuitive handling of heterogeneous data, much like a SQL table or an Excel spreadsheet, but with the added power of Python programming.

Pandas excels in handling various data formats. It can read and write data from multiple sources, including CSV, Excel, JSON, SQL databases, and more. This flexibility makes it invaluable for data ingestion and preparation tasks, often the most time-consuming parts of data analysis projects.

The library offers a rich set of functions for data manipulation. Operations like filtering, sorting, grouping, merging, and reshaping data are streamlined and efficient. Pandas' indexing capabilities are particularly powerful, allowing for complex data selection and transformation operations. The loc and iloc indexers provide intuitive ways to access data by label or position, respectively.

Time series functionality is another strength of pandas. It provides date range generation, frequency conversion, moving window statistics, and date shifting and lagging. These features make pandas particularly useful for financial analysis, scientific research, and any field dealing with time-indexed data.

Pandas integrates seamlessly with other libraries in the Python data science stack. It works well with NumPy for numerical computing, Matplotlib and Seaborn for visualization, and scikit-learn for machine learning. This interoperability allows for end-to-end data science workflows within the Python ecosystem.

The library's handling of missing data is sophisticated, offering various strategies for dealing with NaN (Not a Number) values. Methods for filling, dropping, or interpolating missing data provide flexibility in addressing this common challenge in real-world datasets.

Performance is a key focus of pandas. Many of its underlying operations are implemented in Cython or C, making it efficient for large datasets. Recent versions have introduced nullable integer data types and improved memory usage, further enhancing its capabilities for handling large-scale data.

Pandas also provides powerful tools for data aggregation and analysis. The groupby functionality allows for split-apply-combine operations, essential for summarizing and analyzing data across categories. Pivot tables and cross-tabulations are also supported, enabling complex data reshaping and summarization. One of pandas' strengths is its ability to handle messy, real-world data. Functions for data cleaning, such as removing duplicates, replacing values, and renaming columns, make it easier to prepare data for analysis. The melt and pivot functions allow for easy conversion between wide and long data formats, accommodating different analytical needs.

Recent developments in pandas include improved support for categorical data, enhanced string handling capabilities, and better integration with Arrow for memory-efficient operations. The library continues to evolve, with ongoing efforts to improve performance and expand its functionality.

Pandas has found applications across various industries, from finance and economics to healthcare and social sciences. Its versatility makes it suitable for tasks ranging from simple data cleaning to complex statistical analysis and machine learning model preparation.

4. Numpy

NumPy (Numerical Python) is a fundamental package for scientific computing in Python. Created by Travis Oliphant in 2005, it evolved from earlier packages like Numeric and Numarray. NumPy has become the foundation of the Python scientific computing stack, providing powerful tools for handling large, multi-dimensional arrays and matrices, along with a vast collection of high-level mathematical functions.

At the core of NumPy is the ndarray (n-dimensional array) object, a fast, flexible container for large datasets in Python. Unlike Python's built-in lists, NumPy arrays are homogeneous (all elements must be of the same type) and offer significant performance improvements for numerical operations. This efficiency stems from NumPy's use of contiguous memory blocks and its ability to leverage optimized C code for many operations.

NumPy's array operations are vectorized, meaning they can be applied to entire arrays without explicit loops. This vectorization not only makes code more concise and readable but also significantly boosts performance, especially for large datasets. Operations that might take hundreds of lines in pure Python can often be expressed in a few lines with NumPy, running orders of magnitude faster.

The library provides a comprehensive set of mathematical functions for array operations. These include basic arithmetic operations, trigonometric functions, exponential and logarithmic functions, and statistical operations. NumPy also offers tools for linear algebra, including matrix operations, eigenvalue problems, and solving systems of linear equations.

One of NumPy's strengths is its broadcasting capability. This feature allows operations on arrays of different shapes, automatically replicating smaller arrays across larger ones. Broadcasting simplifies code and improves memory efficiency by avoiding unnecessary copies of data.

NumPy's random number generation capabilities are extensive, supporting various probability distributions. This functionality is crucial for simulations, statistical analyses, and machine learning applications where random sampling is required.

The library also provides tools for reading and writing array data to disk, supporting various file formats. This I/O capability, combined with NumPy's efficient memory usage, allows for handling datasets too large to fit in memory through memory-mapped file arrays.

NumPy's indexing and slicing operations are powerful and flexible. They allow for complex data selection and manipulation, including boolean indexing, which enables sophisticated filtering of data based on conditions.

In the realm of scientific computing, NumPy integrates seamlessly with other libraries. It forms the basis for pandas' DataFrame, is used extensively in SciPy for more advanced scientific computations, and serves as a fundamental data structure in machine learning libraries like scikit-learn and TensorFlow. Recent developments in NumPy have focused on improving performance and expanding functionality. The introduction of generalized universal functions (guifuncs) allows for more flexible vectorized operations. Efforts are also underway to improve NumPy's ability to leverage modern hardware, including better support for GPU acceleration.

NumPy's applications span a wide range of fields, including physics, chemistry, astronomy, geosciences, bioinformatics, and many engineering disciplines. In data science and machine learning, it's often used for data preprocessing, feature engineering, and implementing algorithms from scratch.

The library's documentation is comprehensive, with a wealth of examples and tutorials, making it accessible to both beginners and advanced users. Its large and active community contributes to continuous improvements and provides support through various channels.

NumPy's influence extends beyond Python, serving as a model for array programming libraries in other languages. Its array protocol has been adopted by other Python libraries, ensuring interoperability across the scientific Python ecosystem.

5. Matplotlib:

Matplotlib is a plotting library for Python and its numerical mathematics extension NumPy. Created by John D. Hunter in 2003, Matplotlib has become the foundational library for data visualization in the Python ecosystem. It provides a MATLAB-like interface for creating a wide variety of static, animated, and interactive visualizations.

At its core, Matplotlib offers two primary interfaces: a MATLAB-style state-based interface (pyplot) and an object-oriented interface. The pyplot interface is designed for simple plotting tasks and closely mimics MATLAB's plotting commands, making it easy for users familiar with MATLAB to transition to Python. The object-oriented interface, on the other hand, provides more fine-grained control over plot elements and is better suited for complex visualizations or when embedding plots in graphical user interfaces. Matplotlib's architecture is highly modular, consisting of three main layers: the backend layer (for rendering plots), the artist layer (for representing graphical elements), and the scripting layer (for user interaction). This design allows for flexibility in output formats, including PNG, PDF, SVG, and interactive displays.

The library supports a vast array of plot types, including line plots, scatter plots, bar charts, histograms, pie charts, stem plots, contour plots, 3D plots, and many more. It also provides tools for creating more specialized visualizations like error bars, filled areas, and streamplots. This versatility makes Matplotlib suitable for a wide range of scientific and statistical visualizations.

One of Matplotlib's strengths is its customization capabilities. Users have fine-grained control over virtually every aspect of a plot, from colors and line styles to fonts and axis properties. This level of customization allows for the creation of publication-quality figures tailored to specific requirements. Matplotlib integrates seamlessly with NumPy and can work directly with NumPy arrays. This integration extends to other libraries in the scientific Python ecosystem, such as pandas and SciPy, allowing for easy visualization of data from these sources.

The library also supports LaTeX rendering for mathematical expressions, enabling the inclusion of complex equations and symbols in plot labels and annotations. This feature is particularly valuable in scientific and engineering applications.

Animation support is another key feature of Matplotlib. It provides tools for creating both simple animations and complex, interactive visualizations. This capability is useful for visualizing time-series data, demonstrating algorithmic processes, or creating educational materials.

Matplotlib's event handling system allows for the creation of interactive plots. Users can implement features like zooming, panning, and click events, making it possible to build dynamic, responsive visualizations.

The library is designed with extensibility in mind. It supports custom projections, transformations, and backend systems, allowing advanced users to extend its capabilities for specialized needs.

Recent developments in Matplotlib have focused on improving performance, enhancing 3D plotting capabilities, and providing better support for large datasets. The introduction of style sheets has made it easier to create consistently styled visualizations across projects.

Matplotlib's influence extends beyond direct usage. It serves as the foundation for higher-level plotting libraries like Seaborn, which provides a more user-friendly interface for statistical graphics, and has inspired the design of other visualization libraries in the Python ecosystem.

The library's comprehensive documentation, including a gallery of examples, tutorials, and detailed API references, makes it accessible to users of all levels. Its large and active community contributes to continuous improvements and provides support through various channels.

Matplotlib finds applications across various domains, including scientific research, data analysis, machine learning, financial modeling, and more. Its ability to create a wide range of plot types makes it a versatile tool for visualizing data in fields ranging from physics and astronomy to social sciences and business analytics.

matplotlib

Seaborn :

Seaborn is a statistical data visualization library built on top of Matplotlib and closely integrated with pandas data structures in Python. Created by Michael Waskom in 2012, Seaborn has become a popular choice for creating attractive and informative statistical graphics with minimal code.
At its core, Seaborn aims to make visualization a central part of exploring and understanding data. It provides a high-level interface for drawing attractive and informative statistical graphics, abstracting many of the low-level details that Matplotlib requires. This approach allows users to create complex visualizations with just a few lines of code, making it particularly appealing for data scientists and statisticians.
Seaborn is designed to work seamlessly with pandas DataFrames, leveraging the powerful data manipulation capabilities of pandas to simplify the process of visualization. It can automatically recognize the data types in a DataFrame and choose appropriate plot types and statistical transformations.

6. Plotly:

Plotly is a powerful and versatile graphing library that enables users to create interactive and high-quality visualizations with ease. Developed as an open-source tool, Plotly supports a wide variety of plot types, including basic line and scatter plots, bar charts, histograms, pie charts, and advanced visualizations such as 3D surface plots, heatmaps, and choropleth maps.

One of the key strengths of Plotly is its interactivity. Visualizations created with Plotly allow users to interact with the data through zooming, panning, and hovering over data points to reveal detailed information and tooltips. This interactivity enhances the user experience and helps in exploring and understanding complex datasets more effectively.
Plotly integrates seamlessly with popular programming languages such as Python, R, MATLAB, and JavaScript. This makes it an ideal choice for data scientists, researchers, and developers who want to incorporate sophisticated visualizations into their projects. In Python, for example, Plotly can be used alongside other data analysis libraries such as pandas and NumPy to create comprehensive and dynamic data analysis workflows.
Moreover, Plotly's compatibility with web technologies enables users to share their visualizations easily. Visualizations can be embedded in web applications, Jupyter Notebooks, and dashboards, making it simple to present data insights to a wider audience. Plotly also provides support for exporting visualizations to various formats, including static images and interactive HTML files, ensuring flexibility in how the visualizations are utilized and shared.

Overall, Plotly's combination of powerful plotting capabilities, interactivity, and integration with various programming languages and web technologies makes it a valuable tool for anyone looking to create compelling and insightful data visualizations.

7. Jupyter

Jupyter is an open-source project that provides a powerful environment for interactive computing and data analysis. At its core is the Jupyter Notebook, a web-based application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. This makes it an invaluable tool for data scientists, researchers, educators, and anyone who needs to work with and communicate complex data.

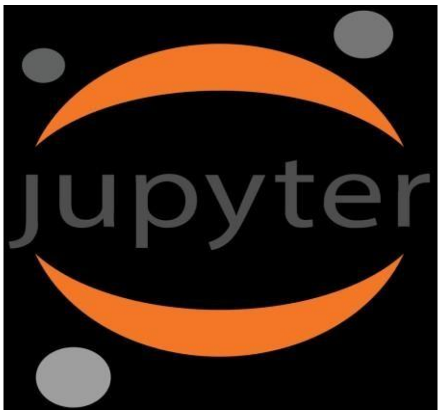
The Jupyter Notebook supports over 40 programming languages, including Python, R, and Julia, allowing for a flexible and versatile coding experience. Users can write and execute code in cells, view the output immediately, and document their thought process and findings within the same document. This combination of code and narrative text promotes a more interactive and exploratory approach to data analysis.

One of the standout features of Jupyter is its ability to integrate with various data visualization libraries, such as Plotly, Matplotlib, and Seaborn, enabling the creation of rich, interactive visualizations within the notebook. This capability is particularly useful for exploring data, testing hypotheses, and presenting results in a clear and engaging manner.

Jupyter Notebooks are widely used in education, as they provide an interactive and hands-on way to teach programming, data science, and other computational topics. They are also extensively used in research and industry for prototyping, data analysis, and sharing reproducible workflows. The notebooks can be easily shared with others, making it simple to collaborate on projects or publish findings.

Additionally, the Jupyter ecosystem includes JupyterLab, a more advanced interactive development environment that extends the capabilities of the traditional notebook interface. JupyterLab offers a more flexible and modular environment, allowing users to arrange multiple documents and activities side by side, such as notebooks, text editors, terminals, and data file viewers.

In summary, Jupyter is a versatile and powerful platform that enhances interactive computing and data analysis. Its ability to combine code, visualizations, and narrative text in a single document fosters a more intuitive and collaborative approach to working with data, making it an essential tool for a wide range of applications.



8. XAMPP

XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends. It consists of Apache HTTP Server, MySQL (or MariaDB) database, and interpreters for scripts written in the PHP and Perl programming languages. The acronym XAMPP stands for Cross-Platform (X), Apache (A), MySQL (M), PHP (P), and Perl (P). It is designed to be easy to install and use, making it an ideal choice for developers looking to create and test web applications locally.

Here are some key components and features of XAMPP:

1. Apache HTTP Server: The Apache server is a widely used web server software that allows users to serve web pages and web applications. It is known for its stability, flexibility, and wide range of features.
2. MySQL/MariaDB: XAMPP includes MySQL or MariaDB, which are powerful relational database management systems. These databases are essential for storing and managing data for dynamic web applications.

3. **PHP:** PHP is a server-side scripting language that is especially suited for web development. It is used to create dynamic web pages that can interact with databases and provide customized content to users.
4. **Perl:** Perl is a highly capable and versatile programming language often used for system administration, web development, network programming, and more. It is included in XAMPP for those who prefer to use it in their web development projects.
5. **Cross-Platform:** XAMPP is available for multiple operating systems, including Windows, macOS, and Linux. This cross-platform compatibility ensures that developers can use XAMPP regardless of their preferred operating system.
6. **Ease of Installation**: XAMPP is designed to be easy to install and configure. The package includes all the necessary components to set up a local web server quickly, making it accessible even for beginners.

7. **Control Panel:** XAMPP comes with a user-friendly control panel that allows users to start and stop the Apache server, MySQL, and other components with a single click. The control panel also provides access to configuration files and logs, making it easier to manage the server environment.
8. **Security**: While XAMPP is intended for local development and testing, it includes several security features to protect the development environment. However, it is not recommended for use as a production server due to its default configuration settings, which prioritize ease of use over security. In summary, XAMPP is a comprehensive and user-friendly tool that simplifies the process of setting up a local web server environment for developing and testing web applications. Its inclusion of essential components like Apache, MySQL/MariaDB, PHP, and Perl, along with its cross-platform compatibility and ease of installation, make it a popular choice among developers.

9. MySQL

MySQL is an open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) for managing and manipulating databases. Developed by Oracle Corporation, MySQL is one of the most popular databases in the world, known for its reliability, ease of use, and performance. It is widely used in various applications, from small-scale projects to large enterprise systems, and is a critical component of many web applications and services.

Key features and characteristics of MySQL include:

1. **Open Source:** MySQL is freely available under the GNU General Public License (GPL), which means users can download, use, modify, and distribute it without cost. This open-source nature has led to a large and active community of developers and users who contribute to its development and improvement.
2. **Relational Database Management System (RDBMS):** MySQL organizes data into tables that can be linked (or related) based on data common to each. This relational model allows for efficient querying and data manipulation using SQL.
3. **SQL Support:** MySQL uses SQL, a standard language for database management, to perform tasks such as querying data, inserting records, updating records, and creating databases and tables. Its adherence to SQL standards ensures compatibility with other SQL-based systems and applications.

4. **Performance and Scalability:** MySQL is designed to handle a wide range of database workloads, from small applications to large-scale enterprise environments. It offers features such as indexing, caching, and optimization techniques to ensure high performance and scalability.
5. **Security:** MySQL includes robust security features to protect data, including user authentication, access control, encryption, and secure connections. It supports SSL/TLS for secure communication between the database server and client applications.
6. **Cross-Platform Support:** MySQL runs on various operating systems, including Windows, Linux, macOS, and Unix-like systems. This cross-platform compatibility makes it versatile and suitable for different development environments.
7. **High Availability:** MySQL supports various high-availability solutions, such as replication, clustering, and failover, to ensure that databases remain accessible and operational even in the event of hardware or software failures.

In summary, MySQL is a robust and versatile RDBMS that offers a powerful combination of performance, scalability, and ease of use. Its open-source nature, wide adoption, and comprehensive feature set make it a preferred choice for developers and organizations looking to manage and leverage their data effectively.

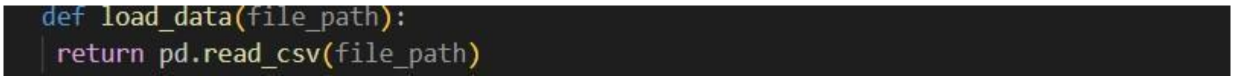
Data visualization

1.Dataset loading:

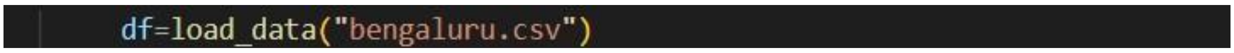
Dataset loading refers to the process of importing data from a source (such as a file, database, or API) into a format that can be used for analysis or machine learning. In Python, popular libraries like pandas are commonly used for loading datasets into a DataFrame, which is a 2-dimensional labeled data structure.

pd.read_csv() is a function in the **pandas** library used to read data from a CSV (Comma-Separated Values) file and convert it into a **pandas DataFrame**. A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure, making it ideal for data manipulation and analysis.

Code :

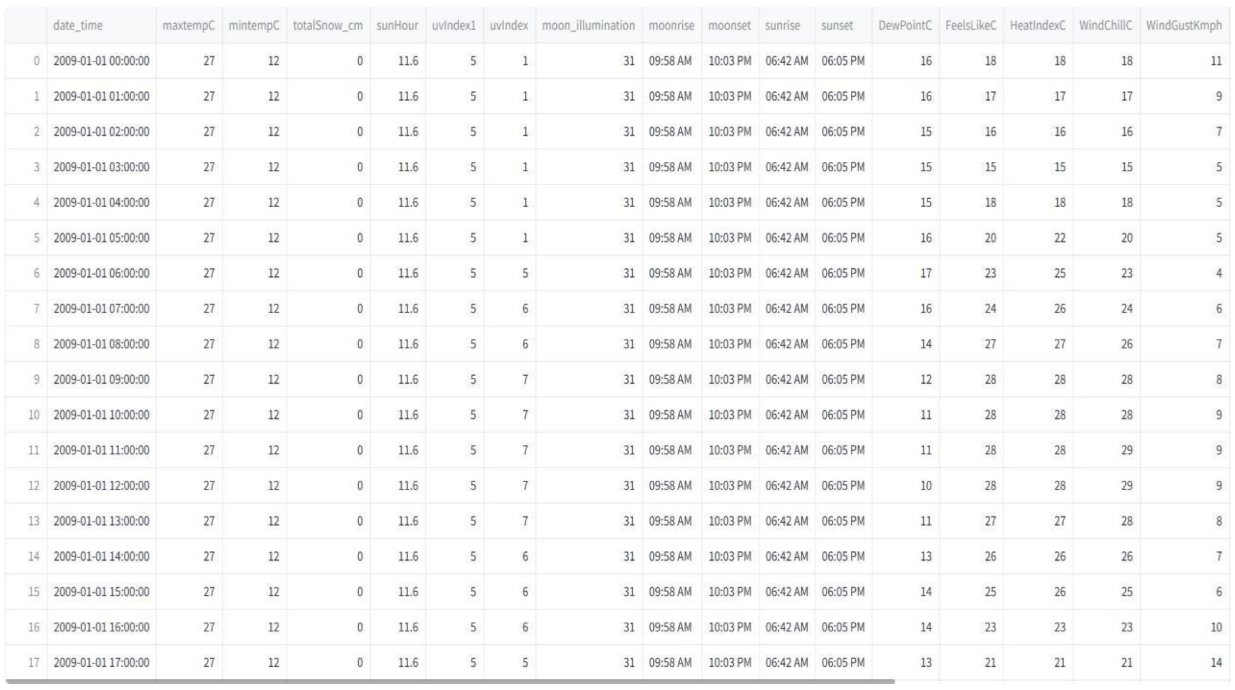


```
def load_data(file_path):
    return pd.read_csv(file_path)
```



```
df=load_data("bengaluru.csv")
```

Output:



	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex1	uvIndex	moon_illumination	moonrise	moonset	sunrise	sunset	DewPointC	FeelsLikeC	HeatIndexC	WindChillC	WindGustKmph
0	2009-01-01 00:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	16	18	18	18	11
1	2009-01-01 01:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	16	17	17	17	9
2	2009-01-01 02:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	15	16	16	16	7
3	2009-01-01 03:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	15	15	15	15	5
4	2009-01-01 04:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	15	18	18	18	5
5	2009-01-01 05:00:00	27	12	0	11.6	5	1	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	16	20	22	20	5
6	2009-01-01 06:00:00	27	12	0	11.6	5	5	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	17	23	25	23	4
7	2009-01-01 07:00:00	27	12	0	11.6	5	6	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	16	24	26	24	6
8	2009-01-01 08:00:00	27	12	0	11.6	5	6	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	14	27	27	26	7
9	2009-01-01 09:00:00	27	12	0	11.6	5	7	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	12	28	28	28	8
10	2009-01-01 10:00:00	27	12	0	11.6	5	7	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	11	28	28	28	9
11	2009-01-01 11:00:00	27	12	0	11.6	5	7	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	11	28	28	29	9
12	2009-01-01 12:00:00	27	12	0	11.6	5	7	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	10	28	28	29	9
13	2009-01-01 13:00:00	27	12	0	11.6	5	7	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	11	27	27	28	8
14	2009-01-01 14:00:00	27	12	0	11.6	5	6	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	13	26	26	26	7
15	2009-01-01 15:00:00	27	12	0	11.6	5	6	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	14	25	26	25	6
16	2009-01-01 16:00:00	27	12	0	11.6	5	6	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	14	23	23	23	10
17	2009-01-01 17:00:00	27	12	0	11.6	5	5	31	09:58 AM	10:03 PM	06:42 AM	06:05 PM	13	21	21	21	14

2.1 Plotting the variation of temperature over years(2009-20019):

The analysis of temperature variation over a decade (2009–2019) provides valuable insights into longterm weather patterns and potential climate trends. By examining the **average maximum temperature** for each year, we aim to understand how the climate has evolved over this period.

1. Overview of the Analysis

Temperature trends over multiple years often reflect changes due to:

- **Natural Climate Variability**: Seasonal cycles, ocean currents, and atmospheric pressure changes.
- **Human Influence**: Urbanization, deforestation, and increased greenhouse gas emissions leading to global warming.

The study focuses on identifying trends in the average maximum temperature across the years to determine whether there is evidence of consistent warming or other noticeable patterns.

2. Observations

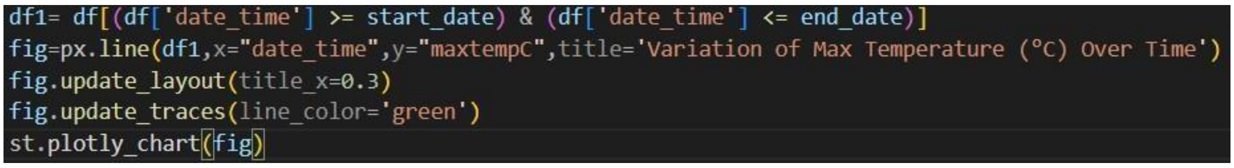
- **Increasing Trend**: If the plot shows a gradual increase in temperatures, it might suggest a warming trend, consistent with global climate change patterns.
- **Year-to-Year Fluctuations**: Some years might show anomalies, such as sudden spikes or dips in temperature, often attributed to events like El Niño or La Niña.
- **Plateaus or Stabilizations**: Certain periods may exhibit relatively stable temperatures, indicating localized or temporary climate conditions.

3. Key Insights

- Over the years 2009–2019, if there is a noticeable upward trend in average maximum temperatures, this could indicate the effects of global warming in the region being studied.
- Anomalies, such as unusually high or low temperatures in specific years, might suggest the influence of extreme weather events, such as heatwaves, storms, or unseasonal rains.

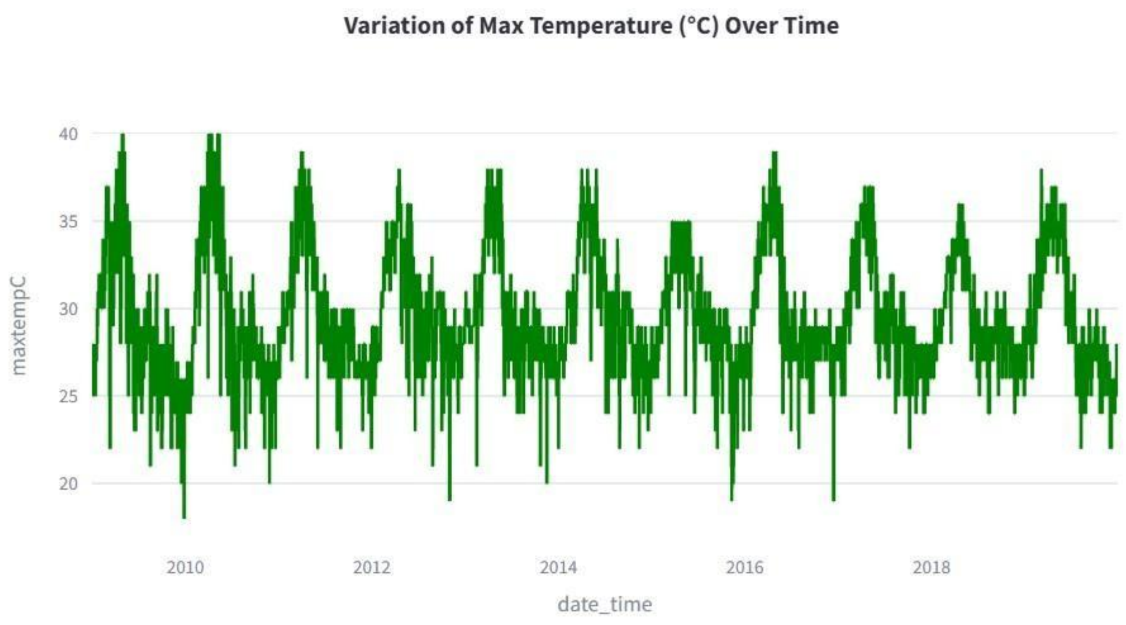
This decade-long analysis serves as a foundation for understanding how local weather patterns align with global climate trends, emphasizing the importance of sustainable practices to mitigate adverse effects.

Code1:

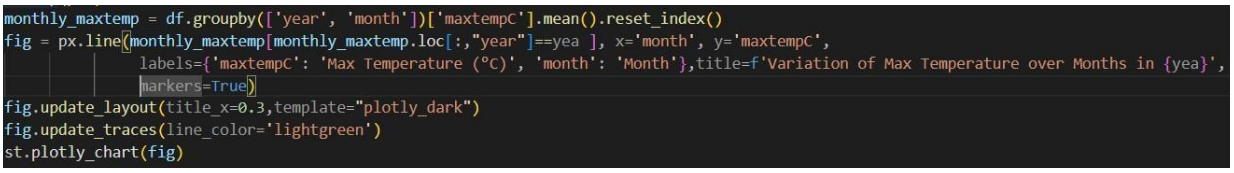


```
df1= df[(df['date_time'] >= start_date) & (df['date_time'] <= end_date)]
fig=px.line(df1,x="date_time",y="maxtempC",title='Variation of Max Temperature (°C) Over Time')
fig.update_layout(title_x=0.3)
fig.update_traces(line_color='green')
st.plotly_chart(fig)
```

Output1:

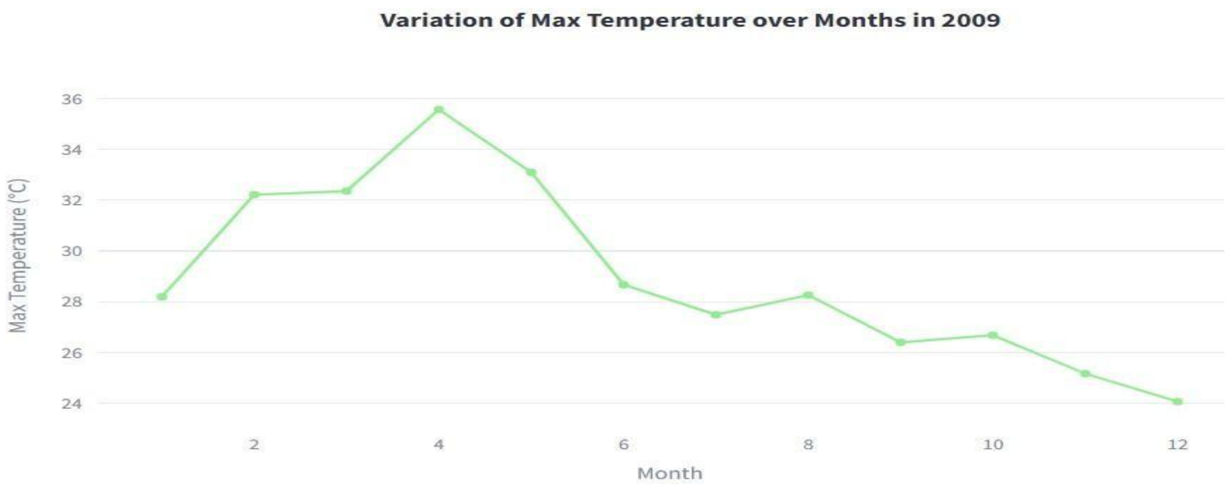


Code 2:



```
monthly_maxtemp = df.groupby(['year', 'month'])['maxtempC'].mean().reset_index()
fig = px.line(monthly_maxtemp[monthly_maxtemp.loc[:,"year"]==yea ], x='month', y='maxtempC',
                labels={'maxtempC': 'Max Temperature (°C)', 'month': 'Month'},title=f'Variation of Max Temperature over Months in {yea}',
                markers=True)
fig.update_layout(title_x=0.3,template="plotly_dark")
fig.update_traces(line_color='lightgreen')
st.plotly_chart(fig)
```

output 2:

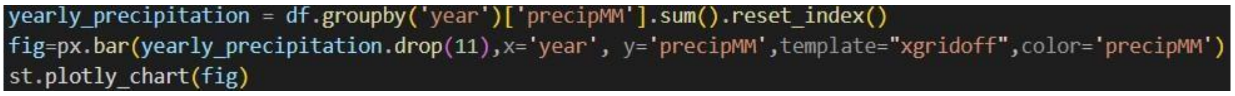


2.2 Plotting the variation of Precipitatiion using bar graph:

Precipitation is a key weather parameter that directly impacts ecosystems, agriculture, and water resource management. By analyzing precipitation over a series of years, we can identify patterns such as wet or dry periods, seasonal variations, and trends that might reflect climate change effects.

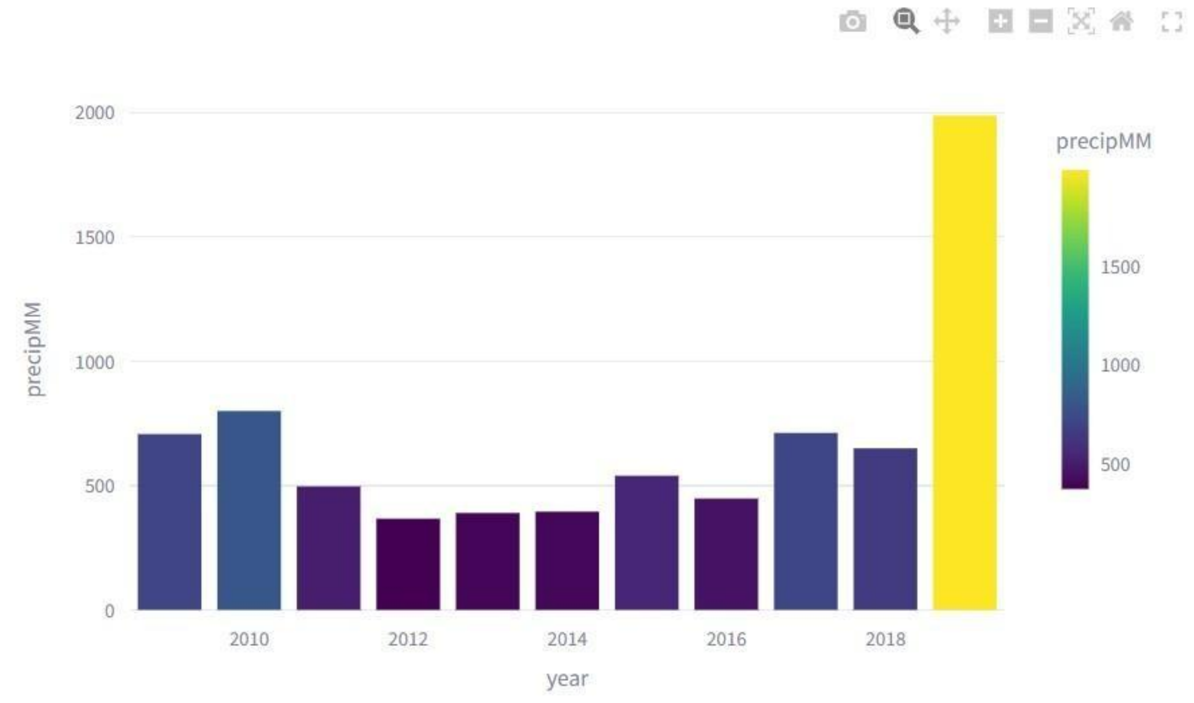
2.2.1 Yearly Precipitaion anlysis: **y**early precipitation is the total amount of rainfall or other forms of precipitation (e.g., snow, hail) recorded within a year. This analysis provides valuable insights into weather patterns and trends, helping us understand climatic changes over time.

Code :



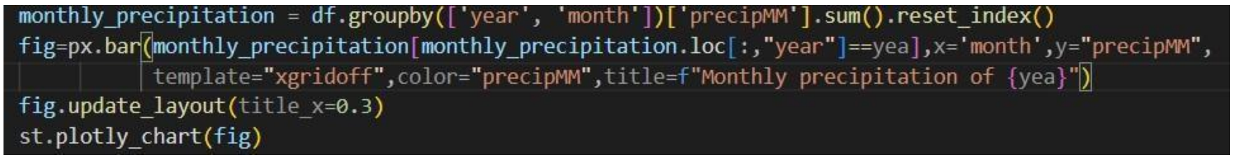
```
yearly_precipitation = df.groupby('year')['precipMM'].sum().reset_index()
fig=px.bar(yearly_precipitation.drop(11),x='year', y='precipMM',template="xgridoff",color='precipMM')
st.plotly_chart(fig)
```

Output:



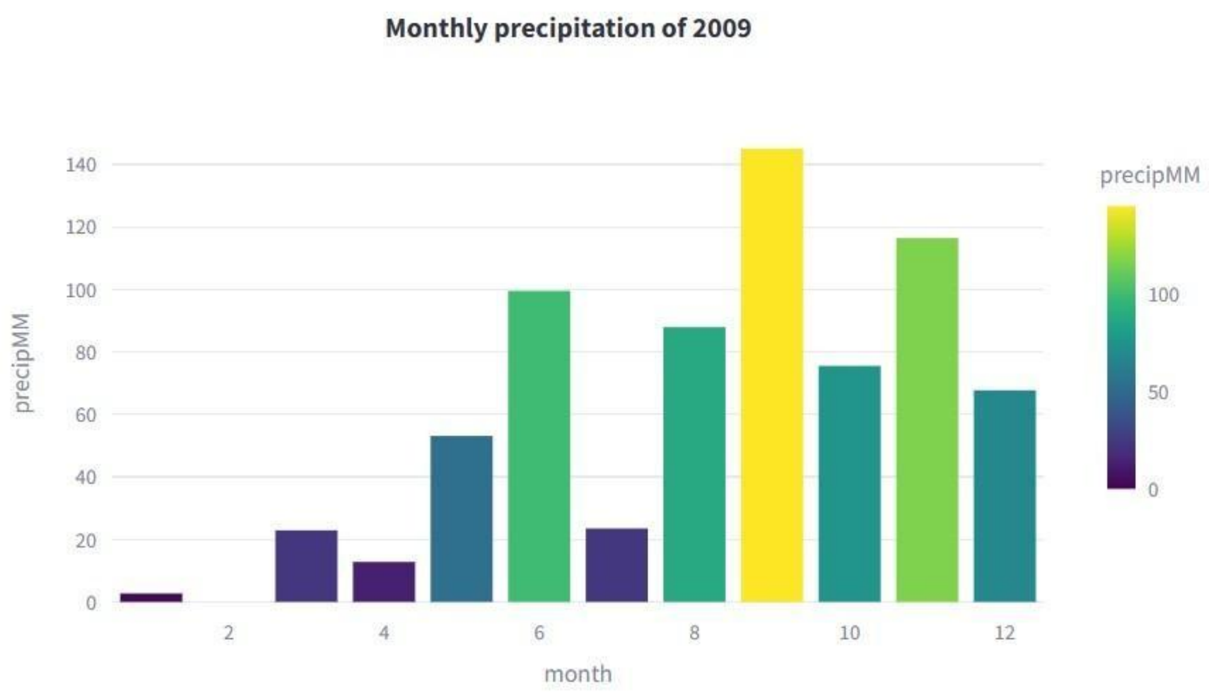
2.2.1 Monthly Precipitation anlysis: Monthly precipitation refers to the total amount of rainfall or other forms of precipitation (e.g., snow, hail) recorded within a specific month. This analysis helps identify seasonal weather patterns and variations, which are crucial for understanding the climate of a region.

Code :



```
monthly_precipitation = df.groupby(['year', 'month'])['precipMM'].sum().reset_index()
fig=px.bar(monthly_precipitation[monthly_precipitation.loc[:,"year"]==yea],x='month',y="precipMM",
                template="xgridoff",color="precipMM",title=f"Monthly precipitation of {yea}")
fig.update_layout(title_x=0.3)
st.plotly_chart(fig)
```

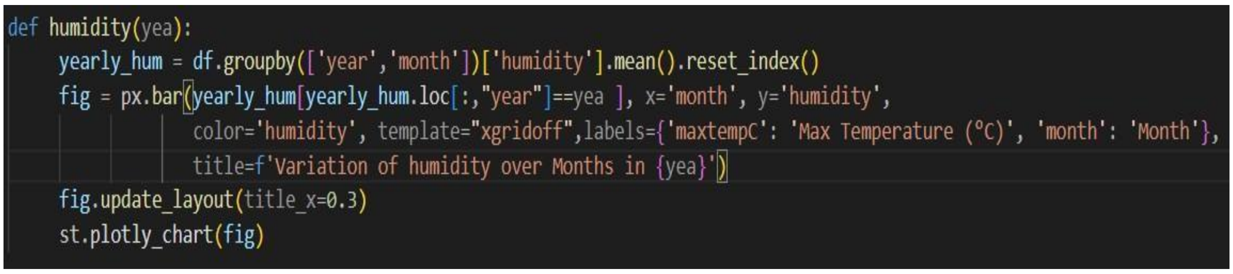
Output:



2.3 Plotting the varaition of Humidity over years(2009-2019):

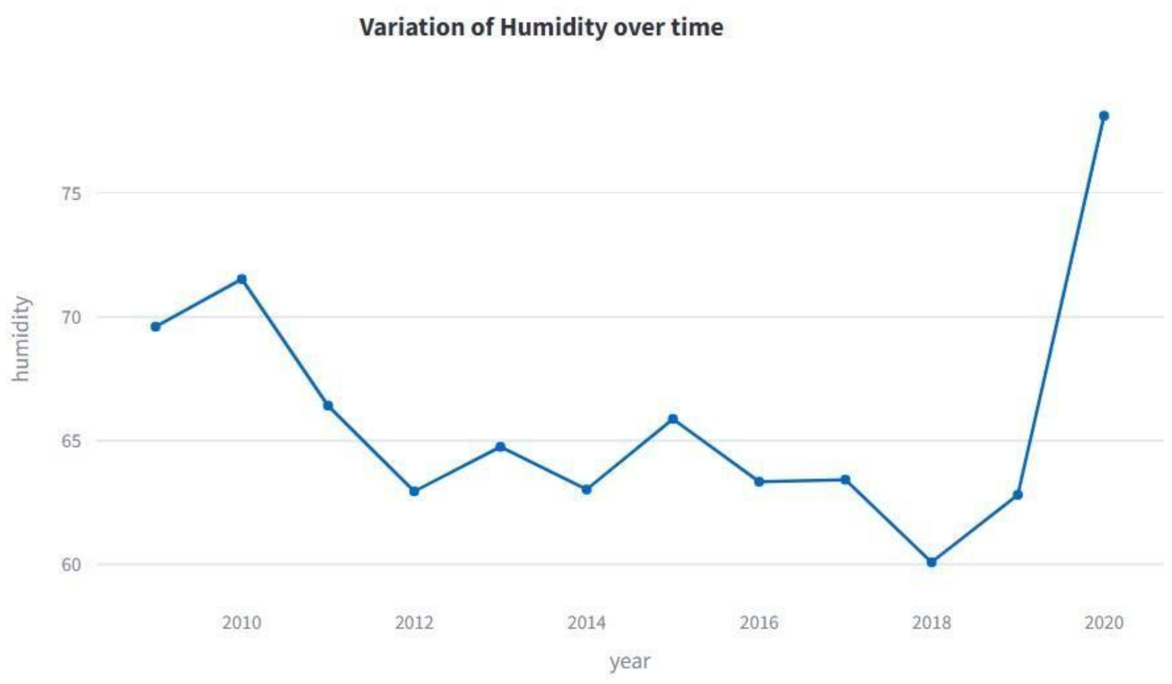
Humidity is a crucial component of atmospheric science, directly influencing weather patterns, climate behavior, and human activities. It is a measure of the amount of water vapor in the air and plays a significant role in temperature regulation, cloud formation, and precipitation. Variations in humidity levels can lead to a wide range of weather phenomena, from dry conditions in deserts to heavy rainfall in tropical regions.

Code:



```
def humidity(yea):
    yearly_hum = df.groupby(['year','month'])['humidity'].mean().reset_index()
    fig = px.bar(yearly_hum[yearly_hum.loc[:,"year"]==yea ], x='month', y='humidity',
                 color='humidity', template="xgridoff",labels={'maxtempC': 'Max Temperature (°C)', 'month': 'Month'},
                 title=f'Variation of humidity over Months in {yea}')
    fig.update_layout(title_x=0.3)
    st.plotly_chart(fig)
```

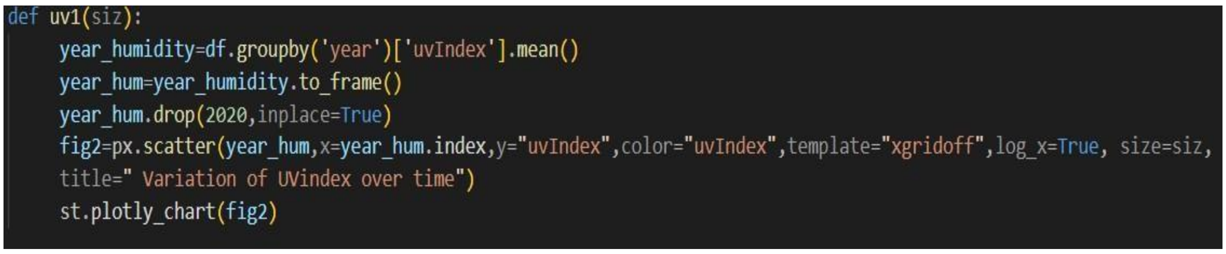
Output:



2.4 Plotting the Variation of UV Index Over Time:

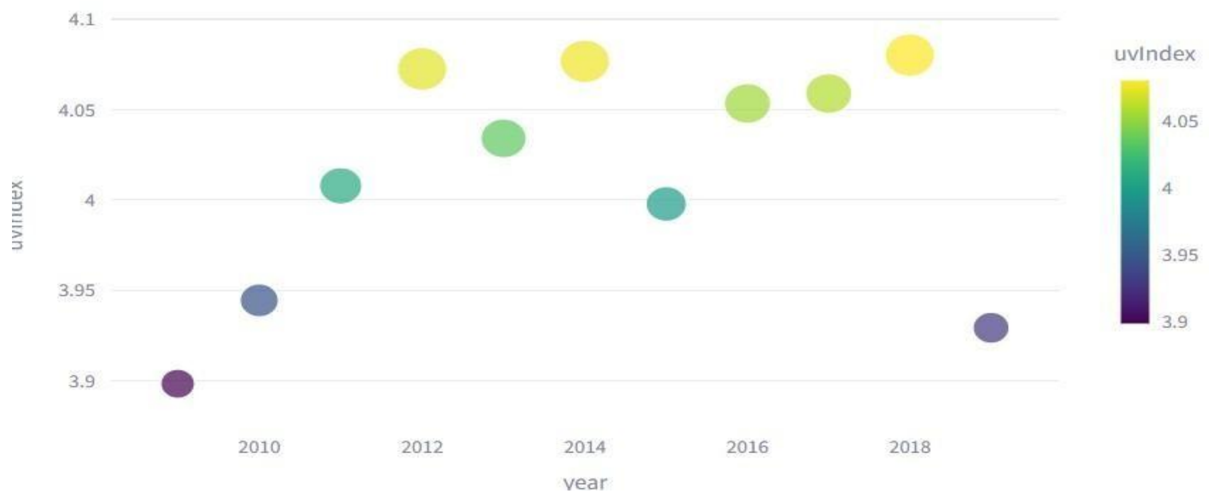
The UV Index is a standardized measure developed by the World Health Organization (WHO) to indicate the strength of ultraviolet (UV) radiation from the sun at a specific location and time. It is a critical tool for assessing the potential risk of overexposure to UV rays, which can cause skin damage, sunburn, and long-term issues such as premature aging and skin cancer. The index ranges from 0 (low risk) to 11+ (extreme risk), with higher values indicating greater potential harm. Factors influencing the UV Index include the angle of the sun, altitude, ozone layer thickness, and cloud cover. Awareness of the UV Index helps individuals take protective measures, such as wearing sunscreen, sunglasses, and protective clothing, or seeking shade to minimize health risks associated with UV exposure.

Code1:

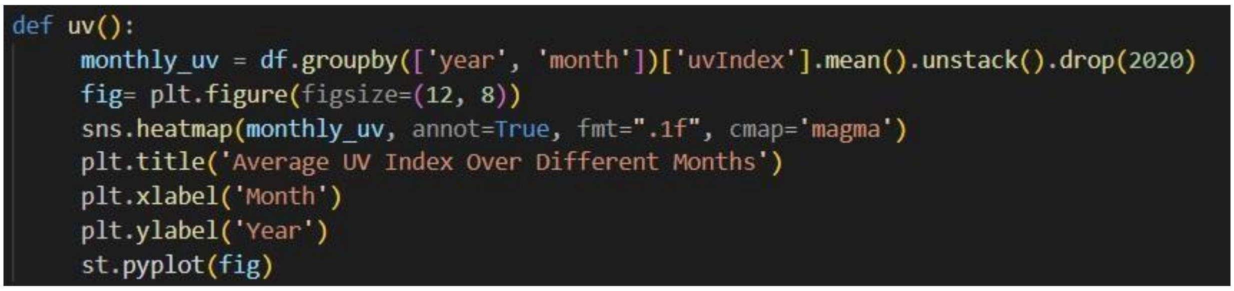


```
def uv1(siz):
    year_humidity=df.groupby('year')['uvIndex'].mean()
    year_hum=year_humidity.to_frame()
    year_hum.drop(2020,inplace=True)
    fig2=px.scatter(year_hum,x=year_hum.index,y="uvIndex",color="uvIndex",template="xgridoff",log_x=True, size=siz,
    title=" Variation of UVindex over time")
    st.plotly_chart(fig2)
```

Output1:

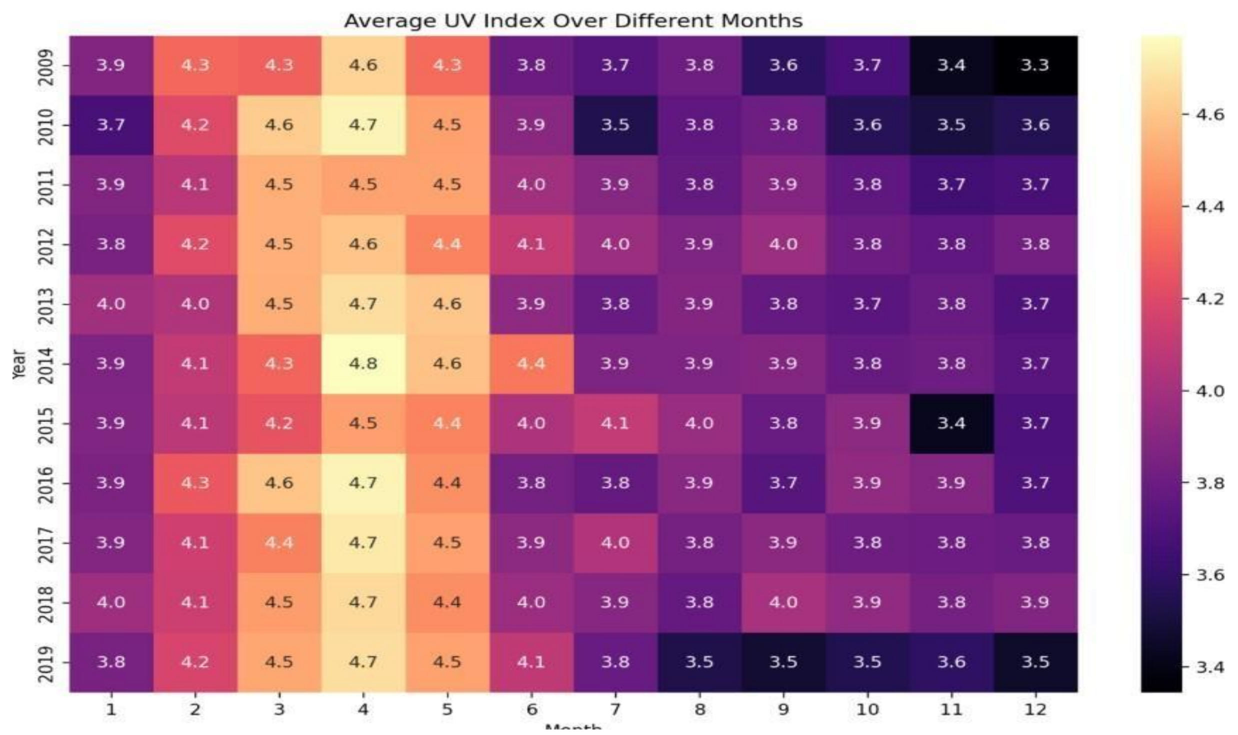


Code2:



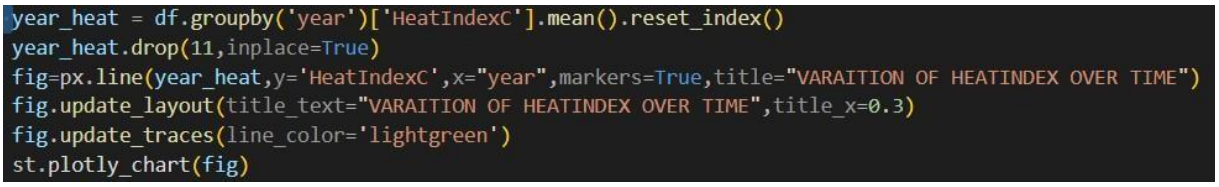
```
def uv():
    monthly_uv = df.groupby(['year', 'month'])['uvIndex'].mean().unstack().drop(2020)
    fig= plt.figure(figsize=(12, 8))
    sns.heatmap(monthly_uv, annot=True, fmt=".1f", cmap='magma')
    plt.title('Average UV Index Over Different Months')
    plt.xlabel('Month')
    plt.ylabel('Year')
    st.pyplot(fig)
```

Output2 :



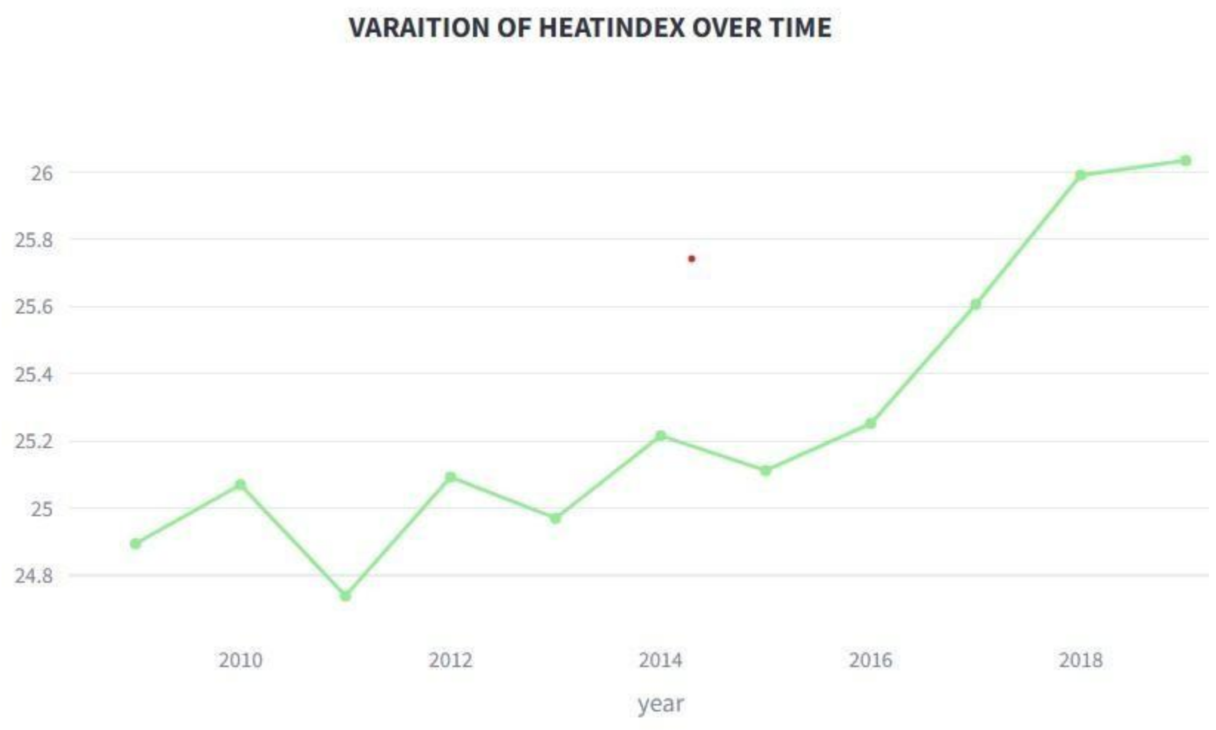
2.5 Plotting the Variation of Heart Index Over Time:

The heat index, often referred to as the "apparent temperature," is a measure of how hot it feels to the human body when relative humidity is combined with the actual air temperature. This index accounts for the reduced ability of the body to cool itself through sweat evaporation in high-humidity conditions, leading to a perceived temperature higher than the actual one. It is commonly expressed in degrees Fahrenheit (°F) or Celsius (°C). The heat index is particularly important for assessing risks associated with heat-related illnesses such as heat exhaustion or heat stroke, as these conditions become more likely when the heat index is high. Values are calculated using a formula or lookup table that considers both temperature and relative humidity. For example, when the temperature is 35°C (95°F) with a relative humidity of 60%, the heat index can feel like 41°C (106°F). This measure is widely used in weather forecasting, health advisories' Code:



```
year_heat = df.groupby('year')['HeatIndexC'].mean().reset_index()
year_heat.drop(11,inplace=True)
fig=px.line(year_heat,y='HeatIndexC',x="year",markers=True,title="VARAITION OF HEATINDEX OVER TIME")
fig.update_layout(title_text="VARAITION OF HEATINDEX OVER TIME",title_x=0.3)
fig.update_traces(line_color='lightgreen')
st.plotly_chart(fig)
```

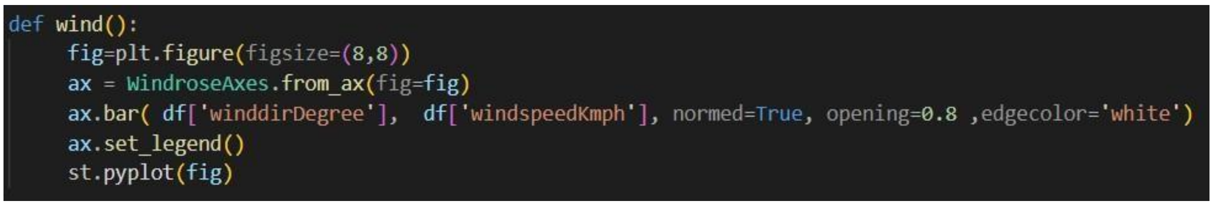
Output:



2.6 **Plotting the Wind Rose Graph Between Wind Speed and Wind Direction**

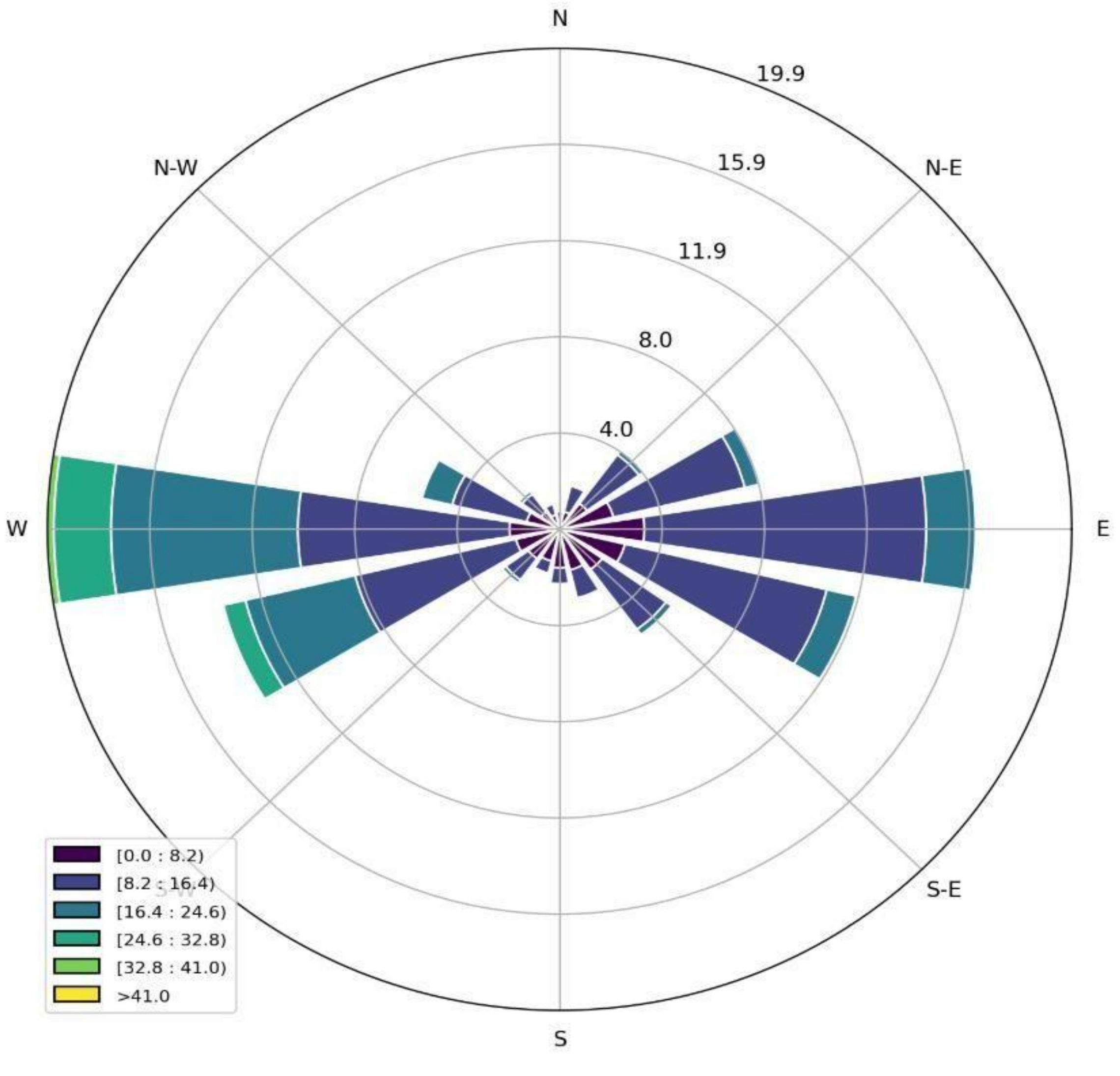
A wind rose graph visually represents the distribution of wind speed and direction, offering a concise summary of how wind varies over time. It is a circular plot where the angles represent wind direction (e.g., North, East, South, West), and the length or color intensity of bars indicates the frequency or magnitude of wind speeds coming from those directions.

Code:



```python
def wind():
    fig=plt.figure(figsize=(8,8))
    ax = WindroseAxes.from_ax(fig=fig)
    ax.bar( df['winddirDegree'],  df['windspeedKmph'], normed=True, opening=0.8 ,edgecolor='white')
    ax.set_legend()
    st.pyplot(fig)
```

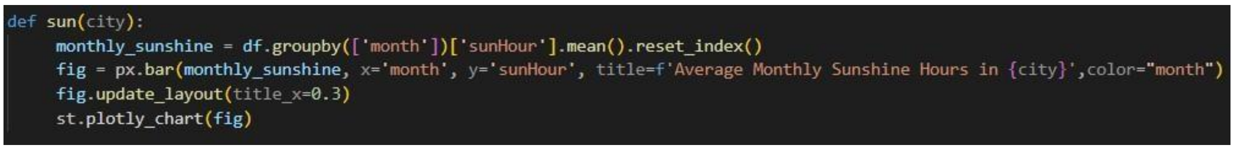
Output:



2.7 Plotting a Heatmap to Show Correlation Between Parameters

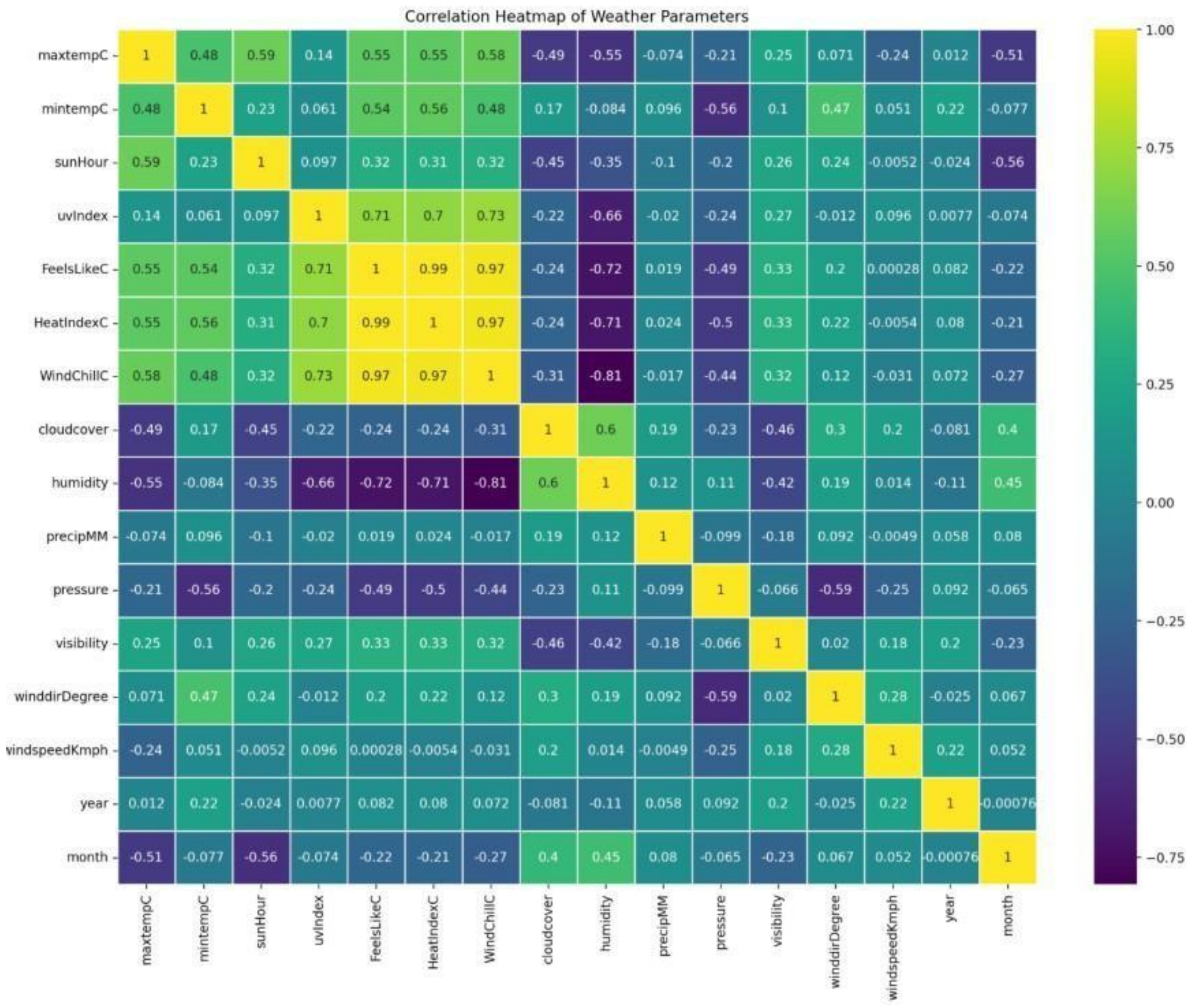
A heatmap is an effective visualization tool for displaying the correlation between different parameters in a dataset. It uses a color-coded matrix to indicate the strength and direction of relationships, where the correlation coefficient ranges from -1 to 1. A value close to 1 indicates a strong positive correlation, -1 indicates a strong negative correlation, and 0 means no correlation.

Code:



```python
def sun(city):
    monthly_sunshine = df.groupby(['month'])['sunHour'].mean().reset_index()
    fig = px.bar(monthly_sunshine, x='month', y='sunHour', title=f'Average Monthly Sunshine Hours in {city}',color="month")
    fig.update_layout(title_x=0.3)
    st.plotly_chart(fig)
```

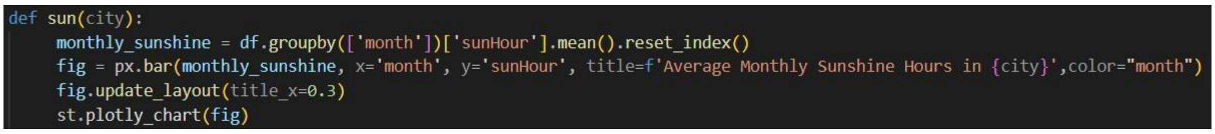
Output:



2.8 Plotting a Bar Graph Between Sun Hours and Years:

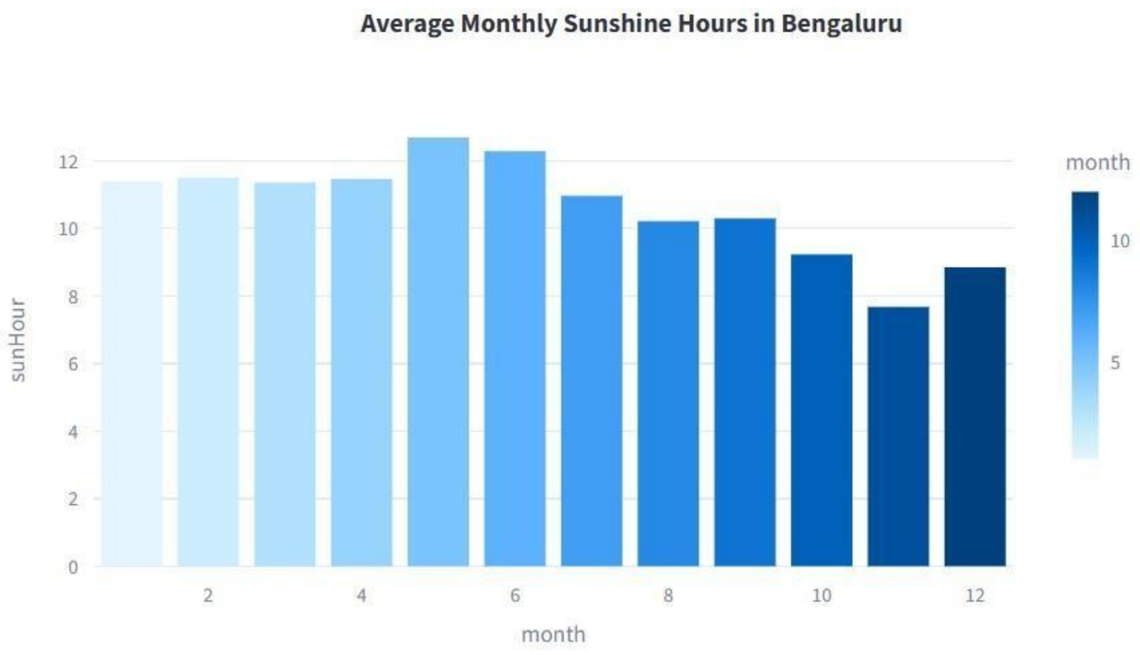
Sun hours, or the duration of sunshine, have a significant impact on various weather parameters. Longer sun hours generally lead to higher temperatures, as the Earth's surface absorbs more solar radiation, causing warming. This increase in temperature also affects humidity levels; warmer air can hold more moisture, which often results in lower relative humidity during the day. Cloud cover tends to be inversely related to sun hours, as more sunshine usually indicates fewer clouds, while cloudy conditions reduce the amount of sunlight reaching the Earth's surface. Furthermore, sun hours can influence wind patterns, as the differential heating of the surface creates pressure gradients, leading to local winds. Extended sun exposure also increases evaporation, contributing to higher moisture levels in the atmosphere, which can eventually lead to rainfall or thunderstorms. Finally, the UV index rises with longer sun hours, heightening the risk of UV-related health issues like sunburn, especially during midday hours when the sun is strongest.

Code:



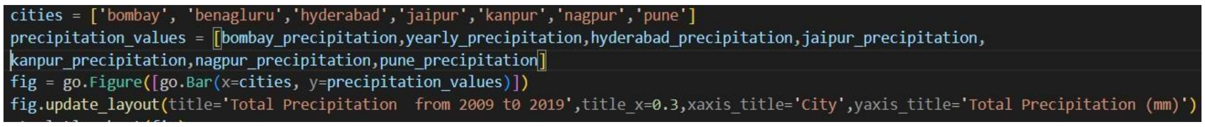
```python
def sun(city):
    monthly_sunshine = df.groupby(['month'])['sunHour'].mean().reset_index()
    fig = px.bar(monthly_sunshine, x='month', y='sunHour', title=f'Average Monthly Sunshine Hours in {city}',color="month")
    fig.update_layout(title_x=0.3)
    st.plotly_chart(fig)
```

Output:



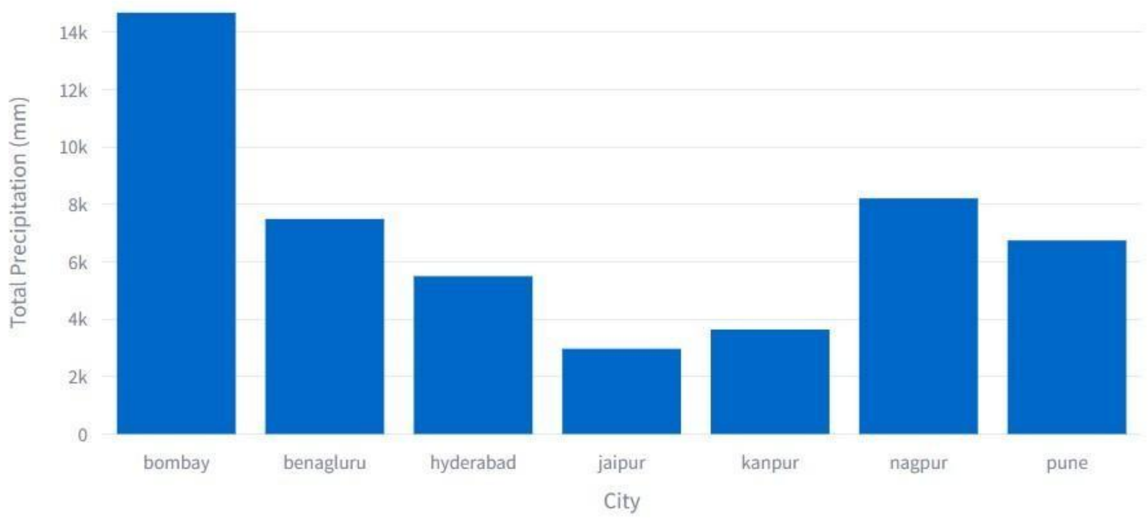
2.9 Compairing the precipitation of different cities over time using bar graph :

Code:



```python
cities = ['bombay', 'benagluru','hyderabad','jaipur','kanpur','nagpur','pune']
precipitation_values = [bombay_precipitation,yearly_precipitation,hyderabad_precipitation,jaipur_precipitation,
kanpur_precipitation,nagpur_precipitation,pune_precipitation]
fig = go.Figure([go.Bar(x=cities, y=precipitation_values)])
fig.update_layout(title='Total Precipitation  from 2009 t0 2019',title_x=0.3,xaxis_title='City',yaxis_title='Total Precipitation (mm)')
```

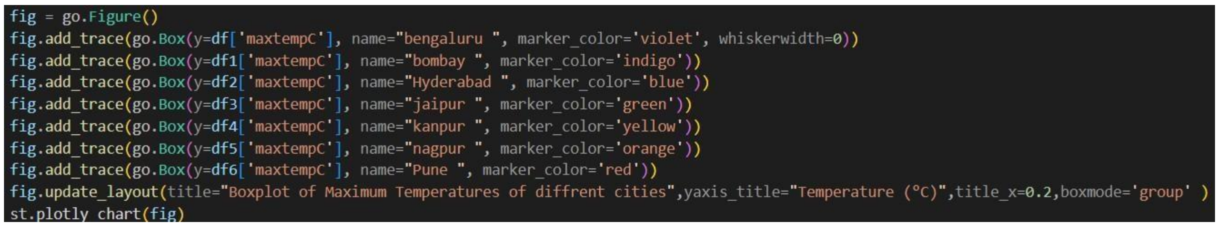
Output:



2.10 Plotting the box plot graph of different cities showing variation in Temperature:

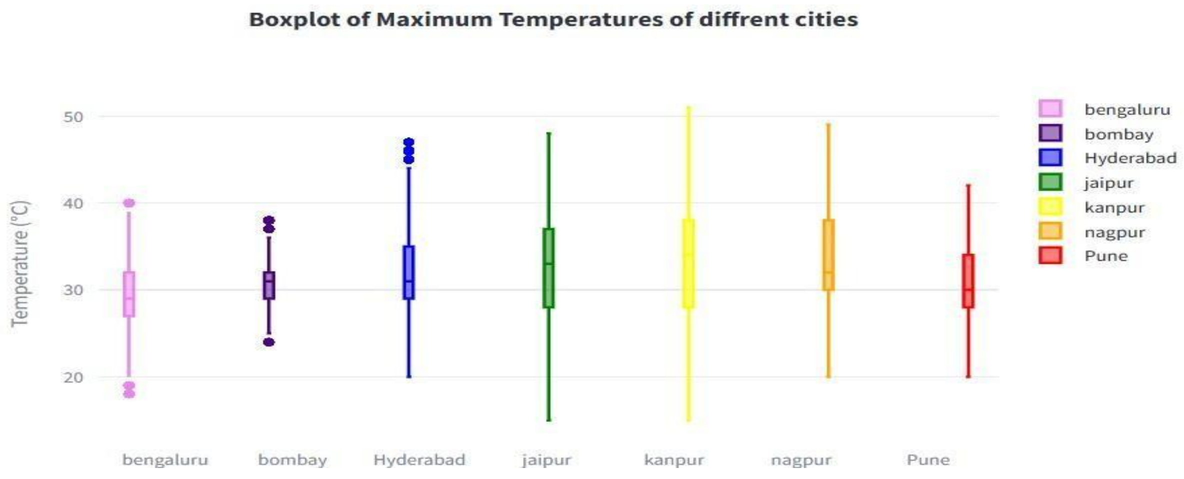
A box plot, also known as a whisker plot, is a statistical visualization tool used to summarize the distribution of a dataset. It highlights key characteristics such as the median, quartiles, and potential outliers. Box plots are particularly useful for comparing the spread and central tendency of data across multiple categories, such as different cities.

Code1:

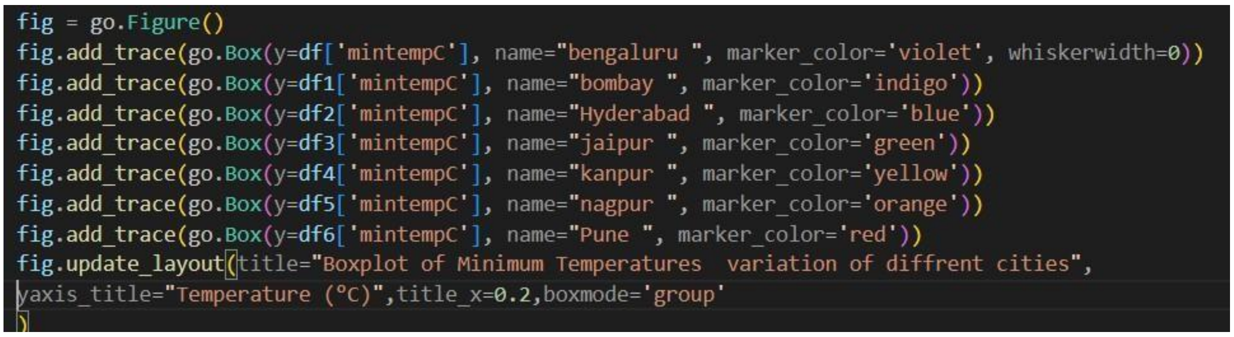


```
fig = go.Figure()
fig.add_trace(go.Box(y=df['maxtempC'], name="bengaluru ", marker_color='violet', whiskerwidth=0))
fig.add_trace(go.Box(y=df1['maxtempC'], name="bombay ", marker_color='indigo'))
fig.add_trace(go.Box(y=df2['maxtempC'], name="Hyderabad ", marker_color='blue'))
fig.add_trace(go.Box(y=df3['maxtempC'], name="jaipur ", marker_color='green'))
fig.add_trace(go.Box(y=df4['maxtempC'], name="kanpur ", marker_color='yellow'))
fig.add_trace(go.Box(y=df5['maxtempC'], name="nagpur ", marker_color='orange'))
fig.add_trace(go.Box(y=df6['maxtempC'], name="Pune ", marker_color='red'))
fig.update_layout(title="Boxplot of Maximum Temperatures of diffrent cities",yaxis_title="Temperature (°C)",title_x=0.2,boxmode='group' )
st.plotly_chart(fig)
```

Output1:

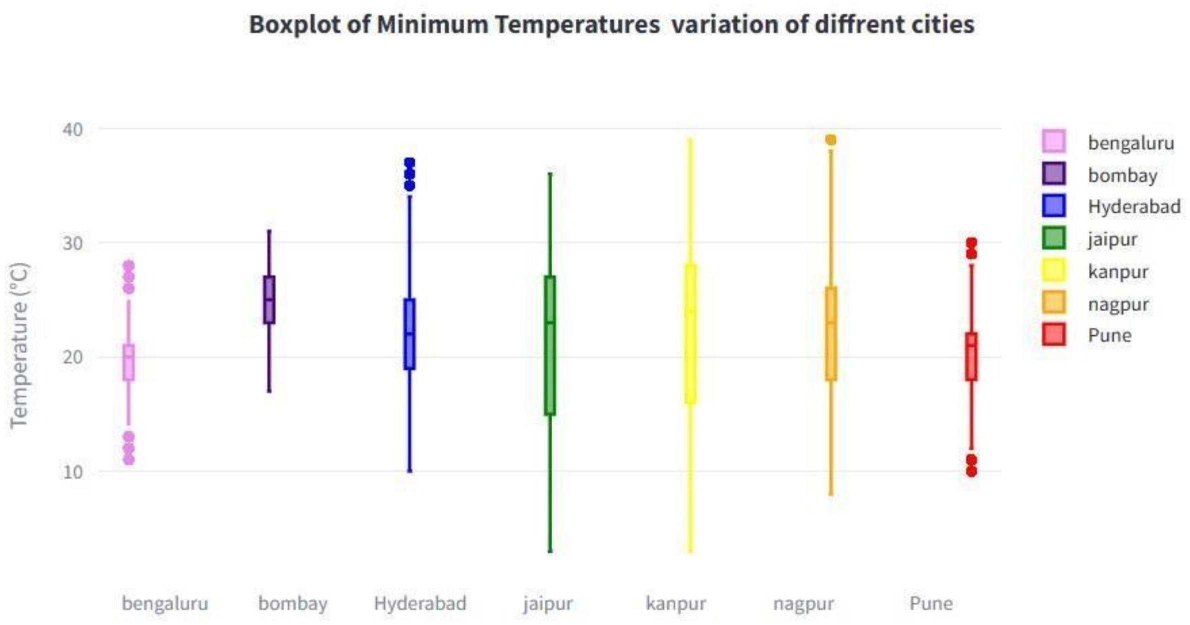


Code2:

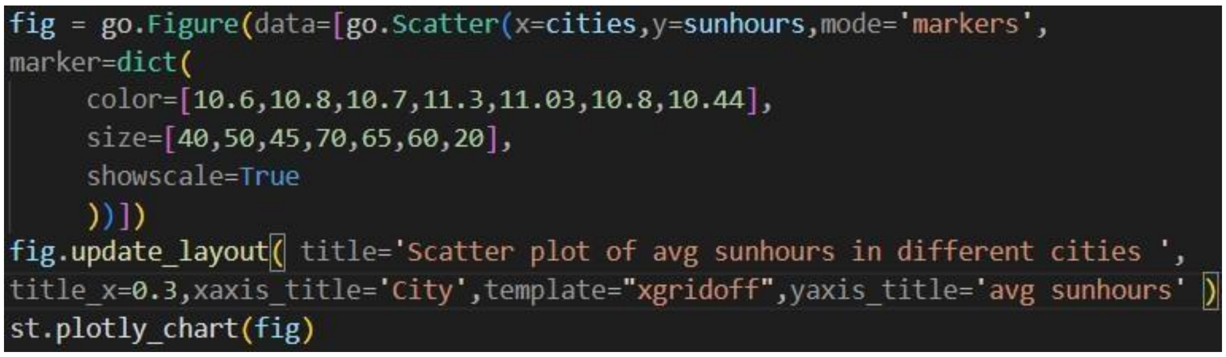


```
fig = go.Figure()
fig.add_trace(go.Box(y=df['mintempC'], name="bengaluru ", marker_color='violet', whiskerwidth=0))
fig.add_trace(go.Box(y=df1['mintempC'], name="bombay ", marker_color='indigo'))
fig.add_trace(go.Box(y=df2['mintempC'], name="Hyderabad ", marker_color='blue'))
fig.add_trace(go.Box(y=df3['mintempC'], name="jaipur ", marker_color='green'))
fig.add_trace(go.Box(y=df4['mintempC'], name="kanpur ", marker_color='yellow'))
fig.add_trace(go.Box(y=df5['mintempC'], name="nagpur ", marker_color='orange'))
fig.add_trace(go.Box(y=df6['mintempC'], name="Pune ", marker_color='red'))
fig.update_layout(title="Boxplot of Minimum Temperatures  variation of diffrent cities",
yaxis_title="Temperature (°C)",title_x=0.2,boxmode='group'
)
```

Output2:

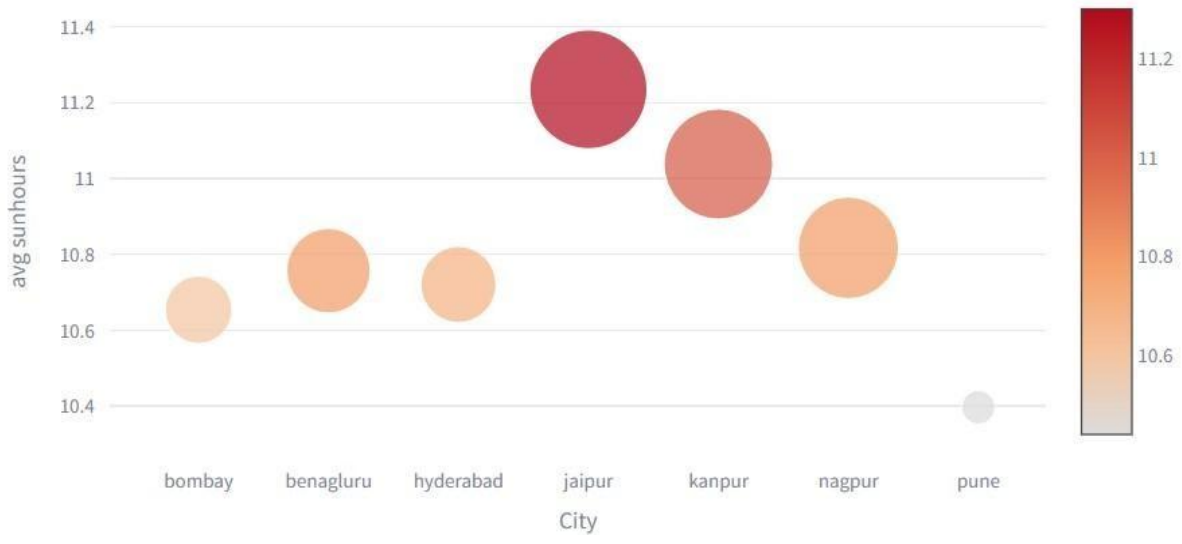


2.11 Plotting the comparison of avg sunhours using scatter plot :
Code:



```
fig = go.Figure(data=[go.Scatter(x=cities,y=sunhours,mode='markers',
marker=dict(
     color=[10.6,10.8,10.7,11.3,11.03,10.8,10.44],
     size=[40,50,45,70,65,60,20],
     showscale=True
     ))])
fig.update_layout( title='Scatter plot of avg sunhours in different cities ',
title_x=0.3,xaxis_title='City',template="xgridoff",yaxis_title='avg sunhours' )
st.plotly_chart(fig)
```

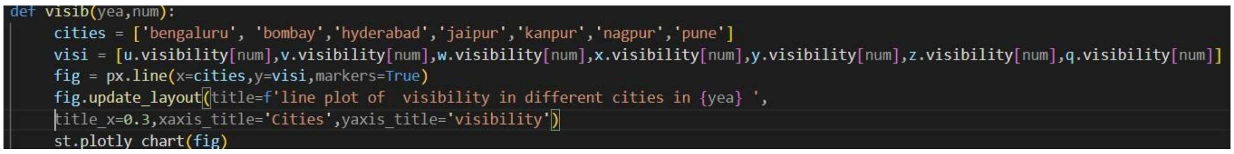
Output:



2.12 Plotting the visibility trends of different cities using line graph:

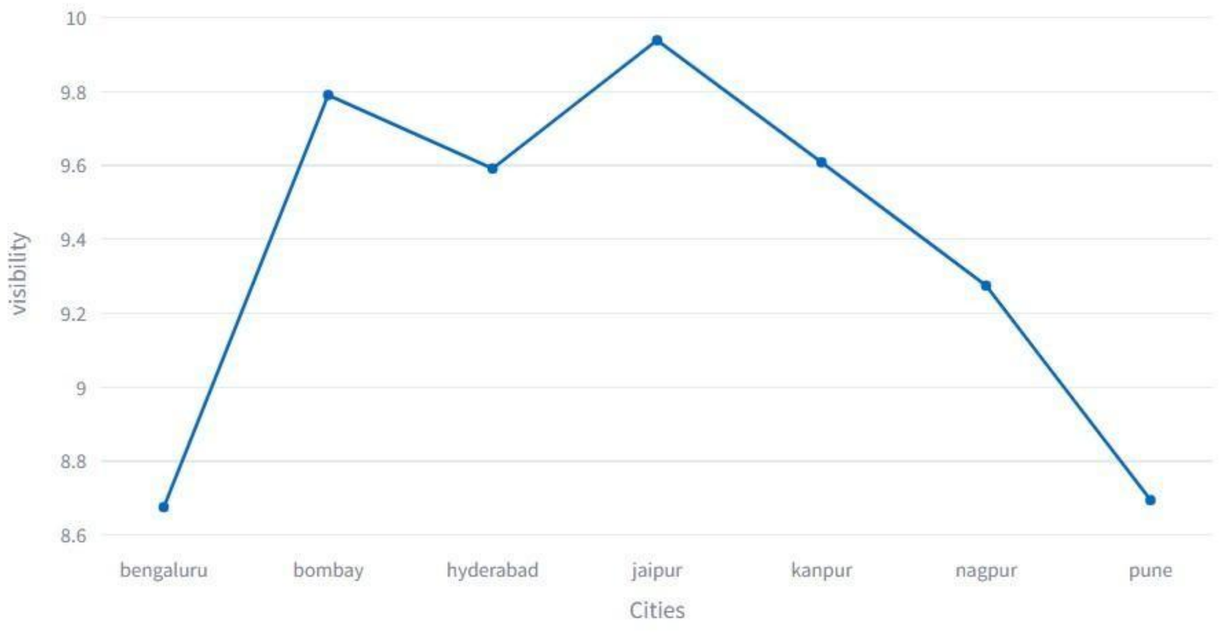
Visibility is a critical meteorological parameter that measures the distance at which an object or light can be clearly discerned by the human eye under prevailing weather conditions.

Code:



```
def visib(yea,num):
    cities = ['bengaluru', 'bombay','hyderabad','jaipur','kanpur','nagpur','pune']
    visi = [u.visibility[num],v.visibility[num],w.visibility[num],x.visibility[num],y.visibility[num],z.visibility[num],q.visibility[num]]
    fig = px.line(x=cities,y=visi,markers=True)
    fig.update_layout(title=f'line plot of  visibility in different cities in {yea} ',
    title_x=0.3,xaxis_title='Cities',yaxis_title='visibility')
    st.plotly_chart(fig)
```

Output:



Interactive GUI Design and Implementation

The Graphical User Interface (GUI), developed using Streamlit, is a user-friendly platform that transforms the static results of weather analysis into an interactive experience. By leveraging the capabilities of Streamlit, the GUI allows users to explore weather trends and correlations dynamically, making the analysis accessible to researchers, policymakers, and the general public, even without technical expertise in data science.

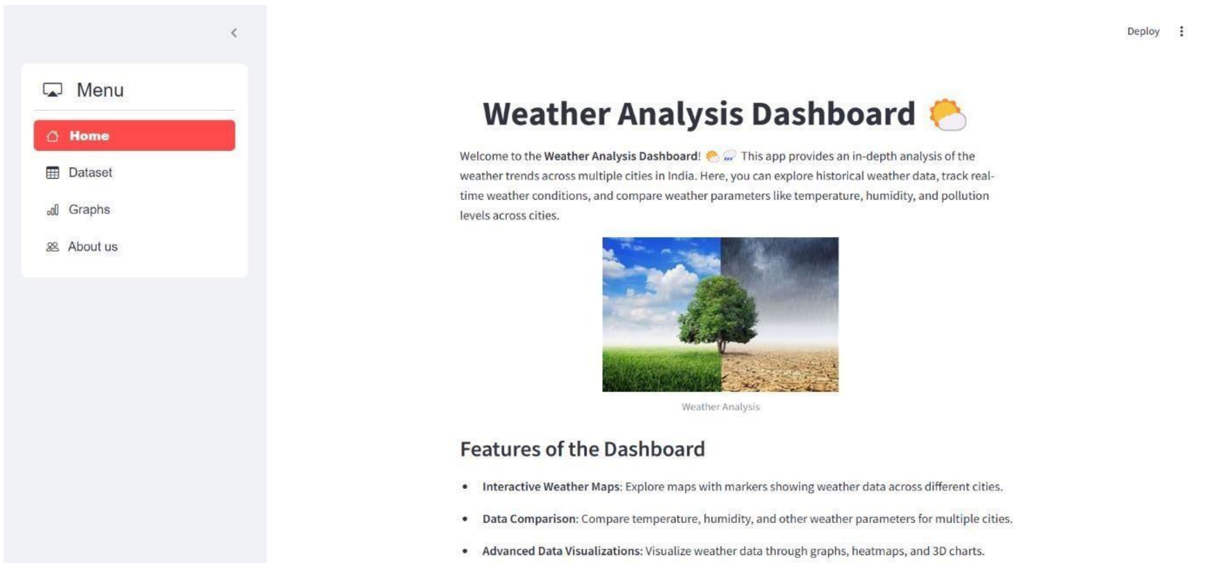
Streamlit's open-source framework provides the foundation for integrating interactive elements like sliders, dropdown menus, and checkboxes. These components enable users to filter data by time periods, weather parameters, and geographic regions, such as Delhi. The GUI features dynamic visualizations created with Python libraries like Matplotlib and Seaborn, allowing users to explore temperature, humidity, and precipitation trends interactively. Its real-time re-execution capability ensures that any change in user input is instantly reflected in the visualizations, delivering a seamless and responsive experience.

The GUI has practical applications in various domains. For instance, it facilitates visibility analysis in urban areas like Delhi, helping stakeholders understand the impacts of air quality on transportation safety. It also supports the exploration of climate trends over time, offering actionable insights for planning and policymaking. By providing a platform for real-time data interaction, the GUI aids in decision-making across fields affected by weather conditions.

Developing the GUI using Streamlit presented challenges, such as optimizing performance for large datasets and ensuring usability for non-technical users. Future enhancements could include integrating machine learning models for predictive analysis, expanding the range of data visualizations, and incorporating additional datasets for more comprehensive insights. Overall, the Streamlit-based GUI bridges the gap between complex data analysis and intuitive user interaction, making the findings of the weather analysis project highly accessible and actionable.

Widgets Used in Different Pages

1. In the **Home Page**, the widgets used include st.markdown for displaying introductory text about the application and its features. It helps organize the content with headings, bullet points, and emphasis. st.image is used to display a weather-related image, adding visual appeal to the page. st.subheader is used to highlight key features of the dashboard, providing a clear structure to the introduction.



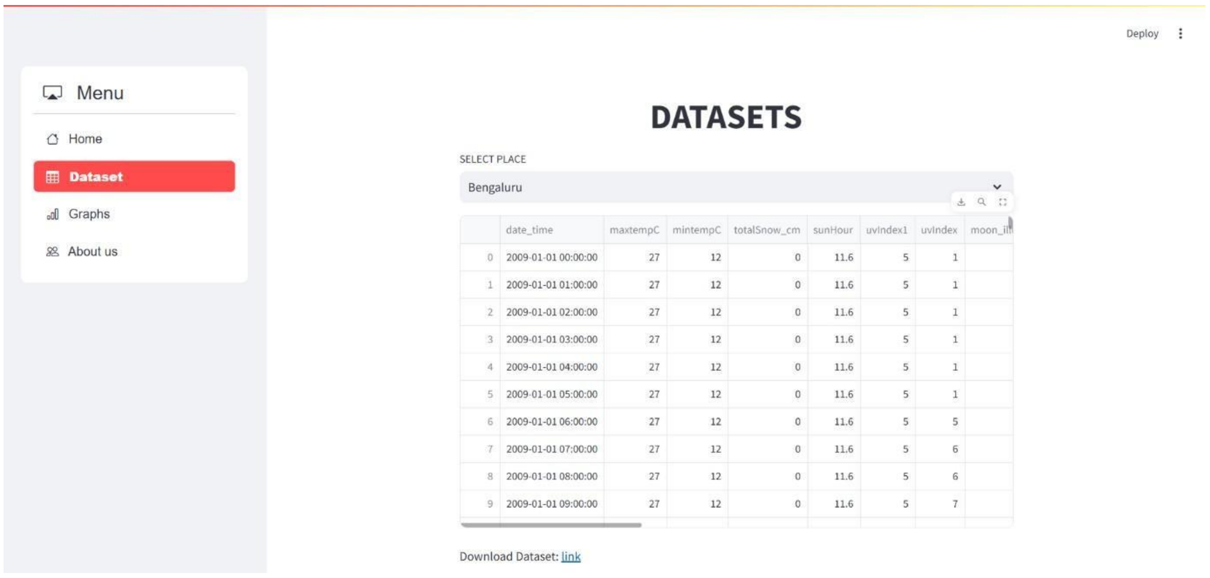
Weather Analysis Dashboard ⛅

Welcome to the **Weather Analysis Dashboard**! ⛅ This app provides an in-depth analysis of the weather trends across multiple cities in India. Here, you can explore historical weather data, track real-time weather conditions, and compare weather parameters like temperature, humidity, and pollution levels across cities.

Features of the Dashboard

- **Interactive Weather Maps**: Explore maps with markers showing weather data across different cities.
- **Data Comparison**: Compare temperature, humidity, and other weather parameters for multiple cities.
- **Advanced Data Visualizations:** Visualize weather data through graphs, heatmaps, and 3D charts.

2.In the **Dataset Page**, a st.selectbox allows users to choose a specific city to view its dataset. Once a city is selected, the dataset is displayed using st.dataframe, which provides a clean, tabular view of the data. Additionally, st.markdown is employed to display a link for downloading the dataset for further analysis.



DATASETS

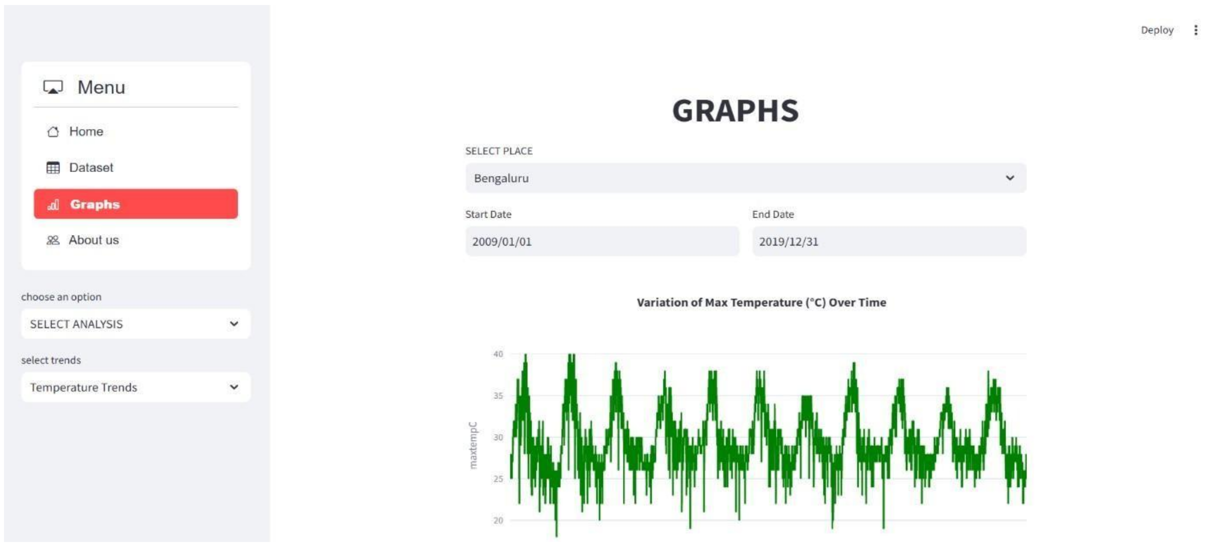
SELECT PLACE

Bengaluru

	date_time	maxtempC	mintempC	totalSnow_cm	sunHour	uvIndex1	uvIndex	moon_ill
0	2009-01-01 00:00:00	27	12	0	11.6	5	1	
1	2009-01-01 01:00:00	27	12	0	11.6	5	1	
2	2009-01-01 02:00:00	27	12	0	11.6	5	1	
3	2009-01-01 03:00:00	27	12	0	11.6	5	1	
4	2009-01-01 04:00:00	27	12	0	11.6	5	1	
5	2009-01-01 05:00:00	27	12	0	11.6	5	1	
6	2009-01-01 06:00:00	27	12	0	11.6	5	5	
7	2009-01-01 07:00:00	27	12	0	11.6	5	6	
8	2009-01-01 08:00:00	27	12	0	11.6	5	6	
9	2009-01-01 09:00:00	27	12	0	11.6	5	7	

Download Dataset: link

3.In the **Graphs Page**, a st.selectbox is used for selecting cities or analysis parameters like temperature trends or precipitation. A st.slider is utilized to select a year or range of years for analysis, enabling precise data filtering. A st.checkbox is available to toggle additional features like correlation analysis, while st.radio allows users to choose between different analysis types, such as city comparisons or specific weather parameters. For visualizations, st.plotly_chart is used to display interactive graphs created with Plotly, and st.pyplot is used for rendering Matplotlib or Seaborn visualizations like wind rose graphs or heatmaps.



choose an option

SELECT ANALYSIS

select trends

Temperature Trends

GRAPHS

SELECT PLACE

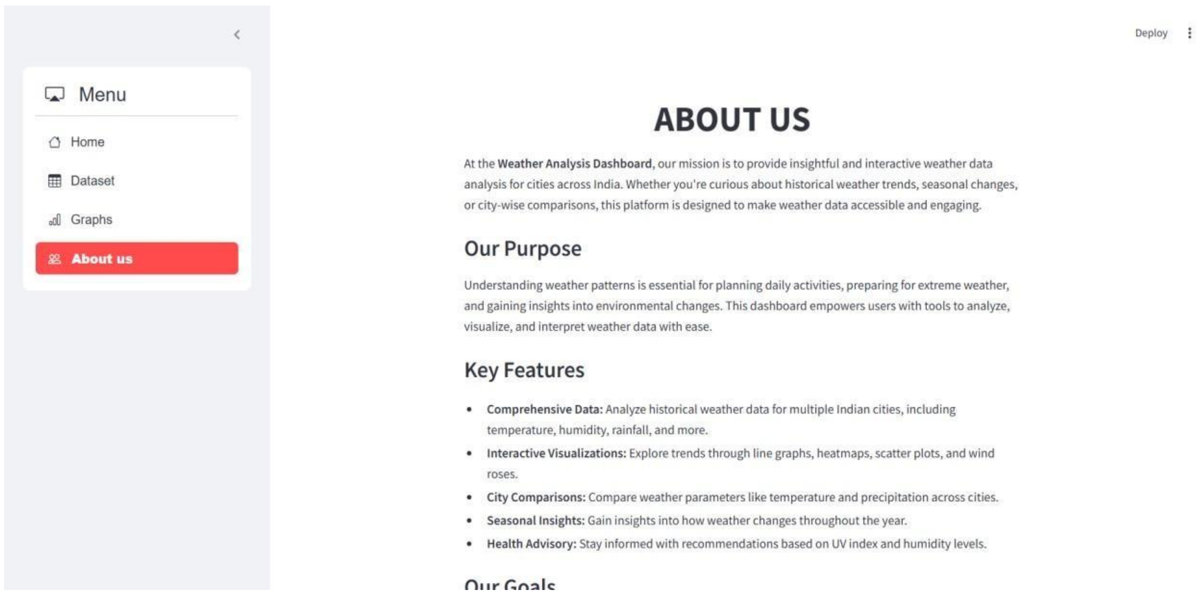
Bengaluru

Start Date 2009/01/01

End Date 2019/12/31

Variation of Max Temperature (°C) Over Time

4.In the **About Us Page**, st.markdown is extensively used to describe the project's purpose, features, and goals, formatted with headings and structured text. Additionally, st.subheader highlights sections within the description, ensuring that the information is well-organized and easy to read.



ABOUT US

At the **Weather Analysis Dashboard**, our mission is to provide insightful and interactive weather data analysis for cities across India. Whether you're curious about historical weather trends, seasonal changes, or city-wise comparisons, this platform is designed to make weather data accessible and engaging.

Our Purpose

Understanding weather patterns is essential for planning daily activities, preparing for extreme weather, and gaining insights into environmental changes. This dashboard empowers users with tools to analyze, visualize, and interpret weather data with ease.

Key Features

- **Comprehensive Data:** Analyze historical weather data for multiple Indian cities, including temperature, humidity, rainfall, and more.
- **Interactive Visualizations:** Explore trends through line graphs, heatmaps, scatter plots, and wind roses.
- **City Comparisons:** Compare weather parameters like temperature and precipitation across cities.
- **Seasonal Insights:** Gain insights into how weather changes throughout the year.
- **Health Advisory:** Stay informed with recommendations based on UV index and humidity levels.

Our Goals

Challenges

The weather analysis project, while providing valuable insights, also presented several challenges:

Data Quality and Incompleteness: One of the major challenges was dealing with missing or incomplete data. In weather datasets, some parameters may be missing for certain hours, days, or months due to sensor malfunctions or other factors. Handling these gaps required careful preprocessing techniques like interpolation or removal of incomplete records, which could affect the accuracy of the analysis.

Data Accuracy: Ensuring that the data used was accurate and reliable was another challenge. Weather data, especially when sourced from different stations or regions, might have inconsistencies or errors. Verifying and cleaning the data was crucial to avoid misleading insights.

Handling Large Datasets: Weather datasets can be large, especially when covering several years of hourly or daily observations. Processing and visualizing such large datasets can slow down the application or even cause memory issues. Optimizing the code and using efficient data handling techniques (like using pandas' read_csv with specific options) was necessary to maintain performance.

Correlation Analysis Complexity: Determining the exact correlations between multiple weather parameters, such as temperature, humidity, and wind speed, can be challenging. While basic correlations were clear, complex relationships that might depend on other variables (e.g., atmospheric pressure, precipitation) required deeper analysis and possibly additional data.

Visualization Limitations: While **Streamlit** is a powerful tool for creating interactive dashboards, its capabilities in terms of advanced visualizations (like 3D plots or highly customized charts) are limited compared to more specialized visualization tools. Some visualizations, like interactive maps or highly detailed plots, could not be implemented as smoothly in Streamlit.

User Interface Design: Ensuring the Streamlit interface was intuitive and user-friendly posed a challenge. It required careful planning of layout and flow to ensure that users could easily filter data, choose parameters, and understand the visualizations. Creating a clean, minimal design while also packing a lot of features in the GUI demanded time and iterative improvements.
Real-time Data Integration: Although the project used historical data, integrating real-time weather data posed a challenge. Real-time data would require an API connection and continuous data updates, which could lead to issues such as inconsistent data formats, delays, or missing updates.

Performance Issues with Large Visualizations: Some of the visualizations, especially heatmaps or scatter plots, could be slow to render with large datasets. Ensuring that these visualizations load quickly and perform efficiently was challenging, and required optimizing the plotting functions or limiting the scope of the displayed data.

Despite these challenges, the project allowed for learning and problem-solving in areas like data preprocessing, visualization optimization, and interface design. Each challenge contributed to the overall enhancement of the weather analysis tool.

Conclusion

The weather analysis dashboard, developed using **Streamlit**, offers a comprehensive and interactive platform for exploring historical weather data. Through the dynamic visualizations and user-friendly interface, the application provides valuable insights into various weather parameters such as temperature, humidity, and precipitation. Users can filter the data by specific time ranges and locations, allowing for a deeper understanding of weather trends and correlations. The platform's ability to visualize these trends over time assists in identifying seasonal patterns, long-term changes, and regional variations, making it a valuable tool for decision-makers in sectors such as urban planning, agriculture, and disaster management.

The development process involved integrating several Python libraries such as **Pandas**, **Plotly**, and **Seaborn** to handle data manipulation and create interactive graphs. The GUI, built with Streamlit, ensures ease of use, making complex weather data accessible to both technical and non-technical users.

Future Scope

While the current version of the application provides essential weather trend analyses, there are several opportunities to enhance its functionality:

- **Real-Time Data Integration**: Incorporating live weather data feeds would allow users to access up-to-date information, enhancing the application's relevance for immediate weather forecasting and real-time decision-making.
- **Predictive Analytics**: By integrating machine learning models, the dashboard could predict future weather trends, offering forecasting capabilities for long-term planning.
- **Expanded Data Coverage**: Integrating more diverse datasets, such as air quality, wind speed, or pollution levels, would provide a more holistic view of environmental conditions.
- **Advanced Visualizations**: Future iterations could include more sophisticated visualizations, such as 3D plots, interactive maps, and heatmaps, for a more immersive user experience.
- **Mobile and Multi-Platform Compatibility**: Optimizing the dashboard for mobile devices and multiple platforms could make it even more accessible to a wider audience, including field researchers and emergency responders.

By addressing these areas, the weather analysis dashboard could evolve into a more robust tool for diverse users, providing not only historical weather data but also predictive capabilities and real-time insights.

References

Streamlit is the framework used for building the interactive web application in this project. The official documentation provides comprehensive details on using Streamlit to create web applications. For more information, you can refer to the Streamlit Documentation, available at https://streamlit.io/docs.

Pandas is used for data manipulation in the project, including filtering and aggregating weather data. The official documentation of Pandas is available at https://pandas.pydata.org/pandas-docs/stable/.

Plotly is used for creating interactive visualizations in the project, especially for line plots and bar charts. The Plotly Documentation can be accessed at https://plotly.com/python/.

Seaborn is used for additional data visualization, particularly for statistical graphics. For more details on Seaborn, refer to its official documentation at https://seaborn.pydata.org/.

The Python programming language is the backbone of the project. The official Python documentation provides the foundational knowledge for understanding and working with Python in data science and web development. You can access the Python Documentation at https://docs.python.org/3/.

If you plan to incorporate real-time weather data, OpenWeather is one of the popular APIs to use for accessing weather data. The OpenWeather API Documentation is available at https://openweathermap.org/api.

Plotly Express is used for simple and effective plotting in the project, especially for line and bar charts. You can learn more about Plotly Express from its documentation at https://plotly.com/python/plotlyexpress/.

If you decide to include machine learning models in the future, this resource can help you understand how machine learning can be applied to predictive weather analytics. A guide to machine learning techniques is provided by Jason Brownlee, available at https://machinelearningmastery.com/.

For understanding weather data analysis and prediction, "Weather and Climate Data Analysis" by E. De Kok (2017) provides valuable insights. This book is published by Cambridge University Press.

If cloud deployment is implemented for scaling the project, AWS provides a range of services that can be useful. You can refer to the AWS Documentation at https://aws.amazon.com/documentation/.