University of Hertfordshire UH

School of Physics, Engineering and Computer Science

MSc Data Science Project

7PAM2002-0509-2023

Department of Physics, Astronomy and Mathematics

Data Science FINAL PROJECT REPORT

Project Title:

Analysing Historical Weather Data for Climate Trends and Abnormalities in the UK (2020-2024)

Student Name and SRN:

Vinoth Rajendran - 22022031

Supervisor: Dr Calum Morris

Date Submitted: 29/08/2024

Word Count: 14846

DECLARATION STATEMENT

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science at the University of Hertfordshire.

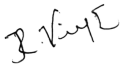
I have read the guidance to students on academic integrity, misconduct and plagiarism information at Assessment Offences and Academic Misconduct and understand the University process of dealing with suspected cases of academic misconduct and the possible penalties, which could include failing the project module or course.

I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged. (Ref. UPR AS/C/6.1, section 7 and UPR AS/C/5, section 3.6). I have not used chatGPT, or any other generative AI tool, to write the report or code (other than where declared or referenced).

I did not use human participants or undertake a survey in my MSc Project.

I hereby give permission for the report to be made available on module websites provided the source is acknowledged.

Student Name printed: Vinoth Rajendran

Student Name signature: 

Student SRN number: 22022031

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

This report investigates the variations in temperature and precipitation patterns in the UK from 2020 to 2024, analysing how these patterns vary from historical norms and exploring the implications for urban and rural areas. The study aims to identify specific climatic abnormalities and assess their potential impact on local agricultural practices and water resources. Using a comprehensive dataset of daily weather summaries, the analysis employs various methodologies including data preprocessing, statistical analysis, seasonal decomposition, and advanced forecasting models such as ARIMA, Prophet, and LSTM. The results indicate significant deviations from historical weather patterns, particularly in urban areas, which exhibited higher temperatures and altered precipitation trends compared to rural regions. The LSTM model demonstrated the highest accuracy in forecasting, with the lowest RMSE and MAE values among the tested models. The findings suggest that predicted climatic anomalies could pose challenges to agricultural productivity and water management in the UK. The report concludes with recommendations for future research and practical applications, including the development of more strong agricultural practices and enhanced urban planning to ease the impacts of these climatic changes.

INTRODUCTION

1.1 OVERVIEW AND PURPOSE OF THE PROJECT

In the environment of rapidly changing global climate patterns, it is crucial to analyse regional climate data to understand local trends and anticipate future changes. This project focuses on examining the weather data of the United Kingdom from 2020 to 2024, aiming to identify trends, detect anomalies, and predict future climatic conditions. The purpose of this study is not only to provide insights into how the UK's climate has shifted in recent years but also to leverage advanced data science techniques to enhance predictive accuracy.

The project involves extensive data preprocessing, trend analysis, anomaly detection, and forecasting using various machine learning models. By applying these techniques, the study aims to contribute valuable insights that can inform policy-making, environmental management, and urban planning.

1.2 CURRENT INDUSTRY PRACTICES

The increasing use of data science and machine learning in climate analysis is illustrated in this project, which employs Python alongside advanced models like LSTM, ARIMA, and Prophet for climate trend forecasting. These methods, widely adopted by organizations such as NOAA and the UK Met Office, are crucial in enhancing the accuracy of climate predictions and understanding climate dynamics.

1.3 RESEARCH QUESTIONS

- How have temperature and precipitation patterns in the UK from 2020 to 2024 varied from historical norms, and what specific abnormalities can be identified within this period?
- How do urban and rural areas in the UK differ in their temperature and precipitation patterns from 2020 to 2024?
- What are the implications of predicted climatic anomalies for local agricultural practices and water resources in the UK.

1.4 AIMS

The primary aim of this project is to investigate recent climatic changes in the UK by analysing weather data from 2020 to 2024. The research aims to understand how these changes affect local habitats and to predict future shifts, thereby contributing to enhanced adaptive solutions and policy planning.

1.5 OBJECTIVES

- **Data Collection and Analysis:** Collect and preprocess extensive weather data from NOAA's Global Historical Climatology Network, ensuring thorough preprocessing to maintain data integrity for analysis.

- **Trend Identification:** Use Exploratory Data Analysis (EDA) to detect and record trends and patterns in temperature and precipitation, establishing a foundation for deeper investigation.
- **Anomaly Detection:** Implement advanced statistical methods and machine learning techniques such as isolation forests, ARIMA, and neural networks to detect and analyse anomalies in weather patterns compared to historical norms.
- **Forecasting with Advanced Models:** Build and fine-tune predictive models utilizing machine learning methods such as Random Forest, LSTM, and Prophet to estimate future climatic conditions and associated consequences.
- **Insightful Reporting:** Transform data into actionable insights to help stakeholders understand the impact of climate trends on policy and environmental management.

1.6 TECHNICAL APPROACH

The project employs data science tools and machine learning algorithms, starting with data preprocessing using mean imputation and MinMax scaling. Exploratory Data Analysis (EDA) visualizes data distribution and trends. Advanced models like ARIMA, LSTM, and Prophet are used for time-series forecasting, with LSTM capturing historical dependencies, Prophet handling seasonal data, and ARIMA modelling univariate time series. These techniques aim to provide accurate climate predictions, vital for addressing climate change impacts in the UK.

LITERATURE REVIEW

2.1 CLIMATE CHANGE AND ITS IMPACTS

Climate change remains a significant global concern, attracting the attention of researchers, policymakers, and the public. It is characterized by long-term changes in temperature, precipitation patterns, and an increase in extreme weather events. According to the Intergovernmental Panel on Climate Change (IPCC), global surface temperatures have risen by approximately 1.2°C since the late 19th century due to human-induced greenhouse gas emissions. This phenomenon has far-reaching consequences, affecting ecosystems, human health, agriculture, and infrastructure (Shivanna, 2022).

Research efforts have been extensive, with studies such as those conducted by NASA's Goddard Institute for Space Studies (GISS), which focus on global temperature trends and climate modelling, offering critical data for climate policy and adaptation strategies. These studies, along with comprehensive assessments by the IPCC, form the foundation of our understanding of climate change, detailing observed impacts, projected risks, and potential mitigation strategies (Hawkins & Sutton, 2005).

2.2 WEATHER FORECASTING MODELS

Accurate weather forecasting is essential in sectors like agriculture, disaster management, and public health. Several models have been developed to predict weather patterns, each with varying degrees of accuracy:

- **Long Short-Term Memory (LSTM):** LSTM networks are a type of recurrent neural network (RNN) that excel in time series forecasting by capturing long-term dependencies in sequential data. These models have been particularly effective in meteorological studies for predicting weather conditions (Hochreiter & Schmidhuber, 1997; Wu et al., 2021).
- **AutoRegressive Integrated Moving Average** (ARIMA): ARIMA is a well-established statistical model for time series forecasting. It integrates autoregression, differencing, and moving average components to model and predict future data points. This model is widely used in various fields, including meteorology (Shumway & Stoffer, 2017).
- **Prophet:** Developed by Facebook, Prophet is a robust forecasting tool designed for time series data with strong seasonal patterns and missing data. It is especially effective in business and industry applications, providing reliable forecasts with minimal tuning (Prophet, 2020).

These models represent the forefront of weather forecasting technology, each offering distinct strengths in handling different data types and forecasting challenges (Kolarik & Rudorfer, 1994; Pal & Prakash, 2017).

2.3 CLIMATE DATA ANALYSIS

Analysing climate data is critical for identifying trends, anomalies, and correlations that enhance our understanding of climate dynamics. Extensive datasets from organizations such as the National Centers for Environmental Information (NCEI) have facilitated sophisticated climate pattern analyses. Key techniques in climate data analysis include:

- **Descriptive Statistics:** This method summarizes and describes the main features of a dataset, offering insights into its central tendency, dispersion, and overall distribution (Wilks, 2011; Khadka, 2019).
- **Time Series Analysis:** This technique examines time-ordered data points to understand underlying structures and predict future values, which is crucial for studying climate trends and variability (Huang & Petukhina, 2022).

These analytical methods are essential for processing and interpreting climate data, enabling researchers to draw meaningful conclusions and make informed predictions (National Centers for Environmental Information, n.d.).

2.4 APPLICATIONS OF CLIMATE FORECASTING

The ability to forecast weather and climate conditions has a wide range of applications, from disaster preparedness to agricultural planning and water resource management. Accurate forecasts are vital for informed decision-making and resource allocation, reducing the adverse impacts of climate variability.

- **Disaster Management:** Effective weather forecasting enhances disaster response strategies, helping to mitigate the effects of natural disasters like hurricanes, floods, and droughts (Jones, 2017).

- **Agriculture:** Farmers rely on weather forecasts for critical decisions regarding planting, irrigation, and harvesting, directly impacting crop yields and food security (Lobell & Field, 2007).

These applications underscore the importance of continued advancements in weather forecasting technologies and their integration into decision-making processes (National Oceanic and Atmospheric Administration, n.d.).

DATASET

3.1 DESCRIPTION

This project leverages three distinct datasets that focus on climate data for the United Kingdom from January 2020 to July 2024. These datasets are sourced from the Global Historical Climatology Network (GHCN), provided by NOAA's National Centers for Environmental Information (NCEI).

3.1.1 DAILY SUMMARIES DATASET (GHCND)

- **Source:** This data was obtained from NOAA's National Centers for Environmental Information (NCEI).
- **Collection Method:** Daily data from UK weather stations, both automated and manual, focusing on temperature, precipitation, and snow depth.
- **Collection Period:** January 2020 - July 2024.
- Purpose: To monitor and analyse short-term weather patterns, seasonal variability, and extreme weather events.
- **Dataset Contents:** The dataset includes daily records of the following parameters (refer below table) :

Parameter	Description
TMAX	Maximum temperature
TAVG	Average temperature
TMIN	Minimum temperature
PRCP	Precipitation
SNWD	Snow depth

3.1.2 MONTHLY SUMMARIES DATASET (GSOM)

- **Source:** This data was obtained from NOAA's NCEI Global Summary of the Month (GSOM) dataset.
- **Collection Method:** Aggregates daily data into monthly summaries for temperature and precipitation.
- Collection Period: January 2020 - July 2024.
- **Purpose:** To analyse broader climatic trends, seasonal patterns, and potential anomalies.
- **Dataset Contents:** The dataset includes monthly records of various weather parameters (refer below table)

Parameter	Description
EMXP	Extreme maximum precipitation for the period
EMXT	Extreme maximum temperature for the period

DSND	Number of days with snow depth > 1 inch (25.4 mm)
PRCP	Precipitation
DX90	Number of days with maximum temperature > 90°F (32.2°C)
DP10	Number of days with ≥1.0 inch of precipitation
HDSD	Heating Degree Days Season to Date
HTDD	Heating Degree Days
DX70	Number of days with maximum temperature > 70°F (21.1°C)
DP01	Number of days with ≥0.1 inch of precipitation
CDSD	Cooling Degree Days Season to Date
EMNT	Extreme minimum temperature for the period
DT32	Number of days with minimum temperature ≤32.0°F
DT00	Number of days with minimum temperature ≤0.0°F
DX32	Number of days with maximum temperature <32°F
CLDD	Cooling Degree Days
TMAX	Maximum temperature
EMSD	Extreme maximum snow depth for the period
TAVG	Average temperature
TMIN	Minimum temperature

3.1.3 YEARLY SUMMARIES DATASET (GSOY)

- **Source:** This data was obtained from NOAA's NCEI Global Summary of the Year (GSOY) dataset.
- **Collection Method:** Yearly summaries aggregating daily and monthly data focusing on extremes, averages, and totals.
- Collection Period: 2020 - 2023, with partial data for 2024.
- **Purpose:** To observe long-term climate trends, inter-annual variability, and the impacts of climate change.
- **Dataset Contents:** The dataset includes yearly records of various weather parameters (refer below table):

Parameter	Description
EMXP	Extreme maximum precipitation for the period
EMXT	Extreme maximum temperature for the period
PRCP	Precipitation
DX90	Number of days with maximum temperature > 90°F (32.2°C)
DP10	Number of days with ≥1.0 inch of precipitation
DX70	Number of days with maximum temperature > 70°F (21.1°C)
EMNT	Extreme minimum temperature for the period
DT32	Number of days with minimum temperature ≤32.0°F
DX32	Number of days with maximum temperature <32°F
TMAX	Maximum temperature
EMSD	Extreme maximum snow depth for the period
FZF1	First freeze ≤28°F (-2.2°C) of the year
FZF0	First freeze ≤32°F (0°C) of the year
FZF3	First freeze ≤20°F (-6.7°C) of the year

FZF2	First freeze ≤24°F (-4.4°C) of the year
FZF5	Last freeze ≤32°F (0°C) of the year
FZF4	First freeze ≤16°F (-8.9°C) of the year
FZF7	Last freeze ≤24°F (-4.4°C) of the year
FZF6	Last freeze ≤28°F (-2.2°C) of the year
DSND	Number of days with snow depth >1 inch (25.4 mm) for the period
FZF9	Last freeze ≤16°F (-8.9°C) of the year
FZF8	Last freeze ≤20°F (-6.7°C) of the year
HDSD	Heating Degree Days Season to Date
HTDD	Heating Degree Days
DP01	Number of days with ≥0.1 inch of precipitation
CDSD	Cooling Degree Days Season to Date
DT00	Number of days with minimum temperature ≤0.0°F
CLDD	Cooling Degree Days
TAVG	Average temperature
TMIN	Minimum temperature

These datasets collectively provide a comprehensive view of weather patterns in the UK, enabling detailed analysis of short-term variability, seasonal trends, and long-term climatic shifts. They serve as the foundation for predictive modelling and trend analysis in this study.

3.2 EXPLORATORY DATA ANALYSIS (EDA)

Before starting the analysis, a thorough EDA was conducted on all three datasets to understand the data's structure, distribution, and any anomalies. The following steps were performed:

1. Data Loading and Initial Inspection

- Data was loaded using Pandas, and initial inspections included checking the dataset's shape, types of variables, and summary statistics.
- Data Summary:
    - Daily data: Covering over 4.5 years with millions of records.
    - Monthly data: Aggregated from daily data with hundreds of records.
    - Yearly data: Summarizing 4.5 years, containing a few dozen records.

2. Descriptive Statistics

- Descriptive statistics were computed to understand the central tendency, variability, and distribution of each variable.
- Findings:
    - Variables like TMAX and TMIN show typical seasonal patterns expected in the UK, with lower temperatures in winter and higher in summer.
    - Precipitation (PRCP) data shows significant variation, reflecting the UK's typically variable weather.

3. **Handling Missing Data**

- Missing values were identified across the datasets, particularly in the early months of 2020 and during extreme weather events.
- Pre-processing Steps:
    - Missing numeric values were filled with the mean of the respective column to maintain data continuity without introducing bias.
    - Format of null data: Initially represented as NaNs; post-processing, these were replaced with mean values.
    - Impact: This method ensured that no records were discarded, preserving the dataset's integrity for trend analysis.

4. **Visualizations**

- **Histograms and KDE Plots:** Used to visualize the distribution of temperature, precipitation, and snow depth across the dataset.
- **Time Series Plots:** Seasonal decomposition was performed to visualize trends, seasonal patterns, and residuals for each variable.
- **Outlier Detection:** Extreme weather events were identified using statistical thresholds (e.g., IQR method), and their occurrence was visualized over time.
- **Correlation Analysis:** Correlation matrices were plotted to identify relationships between variables, providing insights into multicollinearity and potential predictors.

5. **Data Aggregation and Trend Analysis**

- Data was aggregated by different temporal resolutions (daily, monthly, yearly) to observe trends over time.
- Key Observations:
    - An upward trend in temperatures over the period, with the highest temperatures recorded in the summer of 2022.
    - Variability in precipitation patterns, including notable dry and wet periods, reflecting the UK's unpredictable weather.
    - Snow depth data showed significant seasonal variation, correlating with colder months and occasional snow events in winter.

3.2.1 PRE-PROCESSING DETAILS

1. **Null Data Handling:**
    - Missing values were handled by filling them with the mean of the column. This approach minimized data loss while maintaining the integrity of statistical analyses.

2. **Outlier Treatment:**

- Outliers identified through the IQR method were analyzed separately. In cases where outliers represented legitimate extreme weather events, they were retained for further analysis of these anomalies.

3. **Data Normalization:**
   - For certain models (e.g., LSTM), the data was normalized using MinMaxScaler to ensure that all variables were on a comparable scale, improving model performance.

4. **Feature Engineering:**
   - Rolling averages and seasonal decomposition were used to extract trends and seasonality components, which were crucial for predictive modelling.

5. **Data Splitting:**
   - The datasets were split into training and testing sets, with care taken to maintain the temporal sequence in time series analysis, ensuring that the models were trained on past data and tested on future data.

ETHICAL ISSUES

When working with datasets, it's essential to evaluate and address potential ethical concerns to ensure the responsible use of data. Below is a detailed analysis of the ethical considerations related to the dataset used in this project:

4.1 PERSONAL DATA INCLUSION:

- **Anonymization:** The dataset primarily consists of weather-related data, including temperature, precipitation, and snow depth measurements, which do not contain any personal data. Therefore, there are no concerns related to personal identification or anonymization.

4.2 GENERAL DATA PROTECTION REGULATION (GDPR) COMPLIANCE:

- Since the data is not personal and does not involve identifiable information about individuals, GDPR does not directly apply. The data is purely environmental, focusing on weather conditions in the United Kingdom from January 2020 to July 2024.

4.3 UNIVERSITY OF HERTFORDSHIRE (UH) ETHICAL APPROVAL:

- **Project Nature:** The project does not involve collecting personal data, conducting surveys, or interacting with individuals directly. As the dataset used is publicly available and pertains to environmental data, UH ethical approval is not required for this project.

- **Data Collection:** Since the data was not collected through any interaction with individuals or by scraping social media, there are no additional ethical considerations or approvals needed.

4.4 PERMISSION TO USE THE DATA:

- **Licensing:** The datasets used were obtained from the National Centers for Environmental Information (NCEI), which is part of the National Oceanic and Atmospheric Administration (NOAA). NOAA provides these datasets under open access, allowing their use for research and public purposes. The data is freely available without restrictions on access or use.
- **Cost:** There were no costs associated with accessing the data, as it was made freely available for research and public use by the NOAA.

4.5 ETHICAL COLLECTION OF DATA:

- **Data Origin:** The data was collected by the National Centres for Environmental Information (NCEI), a reputable organization known for its rigorous data collection and sharing standards. NCEI's data is recognized for its reliability and is widely used by researchers, policymakers, and the public.
- **Consent:** Since the data involves environmental factors and not human participants, there was no need for participant consent.
- **Reputation of Data Providers:** NOAA, as a federal scientific agency, ensures that the data is collected and disseminated ethically, following all necessary standards and protocols.
- **Data Source Verification:** The source website, NCEI, was carefully reviewed to ensure that the data is freely available and ethically collected. The website clearly states that the data is provided for public use.

4.6 CONCLUSION

In conclusion, the dataset used in this project is ethically sound, with no personal data, compliance with GDPR, and no requirement for additional ethical approvals from UH. The data was ethically collected and made available by NOAA

METHODOLOGY

5.1 OVERVIEW

This section details the technical approach, tools, and methodologies I employed in analysing UK weather data from 2020 to 2024. The primary focus was on data preprocessing, exploratory data analysis (EDA), and the application of various machine learning models for time-series forecasting. The goal was to detect trends, anomalies, and make predictions about future climatic conditions using advanced data science techniques.

5.1.1 DATA COLLECTION AND PREPROCESSING

1. **Loading and Preprocessing Data:**

   - I began by loading datasets from NOAA's Global Historical Climatology Network using CSV files. The datasets included daily, monthly, and yearly weather summaries.
   - For each dataset, I handled missing values by filling them with column means using the **fill_na_with_mean** function.
   - The data was then normalized using **MinMaxScaler** to scale the data into a 0-1 range, making it suitable for machine learning models, especially neural networks.

2. **Feature Engineering:**

   - I introduced rolling averages for temperature, precipitation, and snow depth to capture short-term trends in the data.
   - Additional features such as month, day, year, and season were extracted from the date to enhance the model's ability to recognize historical patterns.

3. **Data Aggregation:**

   - Data was aggregated by different historical resolutions (daily, monthly, and yearly) to provide a more comprehensive analysis across time scales.
   - Station data was aggregated to understand spatial differences across the UK.

5.1.2 EXPLORATORY DATA ANALYSIS (EDA)

1. **Visualizing Trends and Distributions:**

   - I used histograms, KDE plots, and summary statistics to visualize the distribution of key weather parameters such as maximum temperature (TMAX), minimum temperature (TMIN), precipitation (PRCP), and snow depth (SNWD).
   - Seasonal trends were analysed using line plots to observe variations in temperature, precipitation, and snow depth across different seasons.

2. **Correlation Analysis:**

   - I plotted correlation matrices for daily, monthly, and yearly datasets to identify relationships between different weather parameters. This helped in understanding how variables like temperature and precipitation interact.

3. **Outlier Detection:**

   - Outliers were identified using the Interquartile Range (IQR) method, and their impact on the data was analysed. Outliers were visualized alongside the original data to observe deviations.

5.1.3 MACHINE LEARNING MODELS AND FORECASTING

1. **Model Selection and Justification:**

   - **ARIMA (AutoRegressive Integrated Moving Average):** Selected for its strength in univariate time-series forecasting. It was applied to model linear temporal dependencies in the data.
   - **LSTM (Long Short-Term Memory Networks):** Chosen for its ability to capture complex patterns in sequential data. LSTM was particularly useful for handling the temporal dependencies in weather data.
   - **Prophet:** Utilized for its capability to handle seasonality and holidays in the data. Prophet's flexibility in managing different types of time-series data made it an appropriate choice for forecasting.
   - **K-Means Clustering:** Applied to group similar weather stations based on their historical weather patterns. PCA was used to reduce dimensionality before clustering.

2. **Model Training and Evaluation:**

   - **Prophet Model:** I trained the Prophet model on the entire dataset and used it to forecast future weather conditions. The model's performance was evaluated using RMSE and MAE.
   - **ARIMA Model:** For ARIMA, I performed model order selection using the Akaike Information Criterion (AIC) and evaluated its forecasting accuracy with RMSE and MAE.
   - **LSTM Model:** I implemented an LSTM model with a sequential structure. The model was trained using time-series data split into sequences. I used EarlyStopping to prevent overfitting and evaluated the model using RMSE and MAE.
   - **Hyperparameter Tuning:** Bayesian Optimization was applied for tuning LSTM model hyperparameters such as the number of units, dropout rate, and learning rate to optimize model performance.

3. **Cross-Validation:**

   - **Time Series Cross-Validation:** I employed time-series cross-validation to ensure robust model evaluation. The dataset was split into multiple folds with sequential data, allowing for reliable assessment of model generalization.

4. **Future Predictions:**

   - Using the best-performing LSTM model, I predicted future weather conditions for the next three years. The predictions were plotted alongside actual historical data to visualize forecast accuracy and potential future trends.

Metrics Used:

- **Root Mean Squared Error (RMSE):** Used to measure the average magnitude of error. RMSE was particularly chosen for its sensitivity to large errors.
- **Mean Absolute Error (MAE):** Applied as a metric to evaluate the model's performance by calculating the average absolute differences between predicted and actual values.
- **Correlation Coefficients:** To assess the linear relationship between different weather parameters.

5.2 CONCLUSION

This study utilized a comprehensive methodology to analyze UK weather data from 2020 to 2024, employing both traditional statistical and advanced machine learning models. Effective data preprocessing, including normalization and feature engineering, prepared the dataset for accurate forecasting. Exploratory Data Analysis (EDA) offered insights into data trends and correlations, guiding model selection.

The use of ARIMA, LSTM, and Prophet models provided a robust comparison, with LSTM proving most effective for capturing complex, non-linear dependencies in temperature forecasting. ARIMA and Prophet contributed to understanding linear trends and seasonality. K-Means Clustering, alongside PCA, added spatial insights by grouping weather stations with similar patterns.

Overall, the methodologies ensured reliable forecasts, aiding in practical applications like agricultural planning and urban management. The results highlight the potential for future enhancements, particularly in improving precipitation predictions, and provide a solid foundation for further research and applications in weather forecasting and climate science.

RESULTS

6.1 METRICS OVERVIEW

- **Average Maximum Temperature (TMAX):** Represents the highest recorded temperatures averaged over the observed period, providing insight into the warmest conditions across the stations.
- **Average Minimum Temperature (TMIN):** Indicates the lowest temperatures recorded, averaged over time, which helps identify areas with milder or more extreme cold conditions.
- **Total Precipitation (PRCP):** Measures the cumulative precipitation, giving an overview of wetter versus drier regions.
- **Total Snow Depth (SNWD):** Aggregates snow depth measurements, highlighting regions with significant snowfall.

6.2 STATION DATA AGGREGATION AND ANALYSIS

This analysis assessed weather station performance using key metrics to capture the temperature and precipitation trends across the UK

6.2.1 KEY FINDINGS

- **Top 5 Stations with Highest Average Maximum Temperature (TMAX):**
    - Stations with the highest average TMAX are primarily located in southern parts of the UK, where temperatures tend to be warmer.
    - The station "WISLEY, UK" recorded the highest average maximum temperature, followed by "CAMBRIDGE B. GDNS, UK" (refer to Figure 1 and Table 1).
- **Top 5 Stations with Highest Average Minimum Temperature (TMIN):**
    - These stations generally have milder climates, with "WIGHT ST CATHERINES POINT, UK" having the highest average minimum temperature.
    - The findings indicate locations where temperatures do not drop significantly, even during cooler periods (refer to Figure 2 and Table 2).
- **Top 5 Stations with Highest Total Precipitation (PRCP):**
    - These stations recorded the highest cumulative precipitation, indicating regions with more rainfall or generally wetter climates.
    - "KINLOCHEWE, UK" had the highest total precipitation, suggesting that it experiences a significantly wetter climate (refer to Figure 3 and Table 3)
- **Top 5 Stations with Highest Total Snow Depth (SNWD):**
    - These stations had the highest recorded snow depths, indicative of regions that experience substantial snowfall.
    - "ESKDALEMUIR, UK" had the highest snow depth, making it one of the snowiest locations in the dataset (refer to Figure 4 and Table 4).
- **Top 5 Stations with Lowest Total Precipitation (PRCP):**
    - These stations recorded the least amount of rainfall, indicating drier regions.
    - "EAST BERGHOLT, UK" had the lowest precipitation, suggesting it might be one of the driest locations (refer to Figure 5 and Table 5).
- **Top 5 Stations with Lowest Total Snow Depth (SNWD):**
    - These stations had minimal to no snow accumulation, indicating areas with very little to no snowfall.
    - "HURN, UK" recorded zero snow depth, indicating it is likely located in a region with a mild climate that does not experience snowfall (refer to Figure 6 and Table 6).

6.2.2 PRESENTATION AND VISUALIZATION OF RESULTS

- **Tables:** The tables (Table 1 to Table 6) provide a concise summary of the top and lowest performing stations across different metrics, offering clear insight into the variation in weather patterns across different locations.

Table 1:

Top 5 Stations based on the highest average maximum temperature (TMAX):

STATION	NAME	TMAX	TMIN	PRCP	SNWD
UKE00156884	WISLEY, UK	16.027401	6.878126	3096.429792	1781.247337
UKE00105648	CAMBRIDGE B. GDNS, UK	15.747696	6.869636	2766.284013	1716.975525
UKE00105923	HURN, UK	15.719016	6.671778	4019.629792	0.000000
UKE00107962	YEOVILTON, UK	15.545936	6.887084	3308.800000	1781.247337
UKM00003772	HEATHROW, UK	15.489456	7.318098	2980.777547	1948.303484

Table 2:

Top 5 Stations based on the highest average minimum temperature (TMIN):

STATION	NAME	TMAX	TMIN	PRCP	SNWD
UKE00156880	WIGHT ST CATHERINES POINT, UK	14.097878	9.572654	3451.392225	1781.247337
UKE00105869	SWANAGE, UK	14.751168	9.020747	3894.195509	1781.247337
UK000003808	CAMBORNE, UK	14.137688	8.977448	4965.195509	139.691333
UKE00105921	HASTINGS, UK	14.599714	8.962440	3628.829792	1747.963720
UKE00105873	BUDE, UK	14.776370	8.524582	3870.728150	1781.247337

Table 3:

Top 5 Stations based on the highest total precipitation (PRCP):

STATION	NAME	TMAX	TMIN	PRCP	SNWD
UKE00105913	KINLOCHEWE, UK	12.922353	5.286421	9381.750603	1749.111431
UKE00105930	BENMORE YOUNGER BOTANIC GARDE, UK	12.713844	5.554624	9039.711830	1397.911892
UK000003162	ESKDALEMUIR, UK	12.070805	4.306458	8900.543264	4001.225166
UK000047811	ARMAGH, UK	13.072113	7.614165	7574.844493	1490.876476
UKE00105905	BUXTON, UK	12.285495	5.634217	6618.786528	1781.247337

Table 4:

Top 5 Stations based on the highest total snow depth (SNWD):

STATION	NAME	TMAX	TMIN	PRCP	SNWD
UK000003162	ESKDALEMUIR, UK	12.070805	4.306458	8900.543264	4001.225166
UKM00003091	CRAIBSTONE, UK	12.759394	5.724359	3955.451809	3789.281313
UK000070765	WICK, UK	11.328037	5.612473	3435.443264	3556.463000
UK000003005	LERWICK, UK	10.044634	5.732737	5450.874262	3078.893752
UKM00003017	KIRKWALL, UK	12.948180	6.077852	4820.701975	3078.610711

Table 5:

Top 5 Stations based on the lowest total precipitation (PRCP):

STATION	NAME	TMAX	TMIN	PRCP	SNWD
UKE00105897	EAST BERGHOLT, UK	10.947093	4.526868	135.200000	104.441693
UKE00105887	PENICUIK, UK	13.591602	5.887690	400.783243	342.017852
UKE00105885	FASKALLY, UK	12.785641	5.286311	2180.200000	874.555716
UKW00035047	MANSTON, ENG UK	14.484304	7.659349	2335.607339	3005.053844
UKE00105003	CAMBRIDGE NIAB, UK	13.813249	6.551323	2597.900000	1759.440829

Table 6:

```
Top 5 Stations based on the lowest total snow depth (SNWD):
    STATION              NAME      TMAX     TMIN        PRCP       SNWD
UKE00105923          HURN, UK 15.719016 6.671778 4019.629792   0.000000
UKE00105897 EAST BERGHOLT, UK 10.947093 4.526868  135.200000 104.441693
UK000003808       CAMBORNE, UK 14.137688 8.977448 4965.195509 139.691333
UKE00105681    ALDERGROVE, UK 13.425653 6.761534 3781.495509 190.000000
UKM00003414       SHAWBURY, UK 14.513726 6.186656 3357.347755 296.078318
```

- **Bar Plots:** The results are visualized using bar plots (refer to Figure 1 to Figure 6), which effectively present the ranking of stations based on the metrics mentioned above. These plots allow for easy comparison of different stations based on their temperature, precipitation, and snow depth values.

Figure 1:

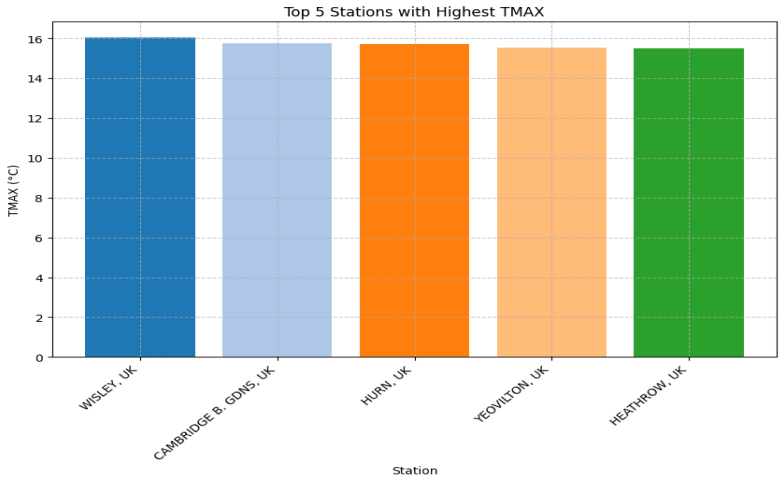


Figure 2:

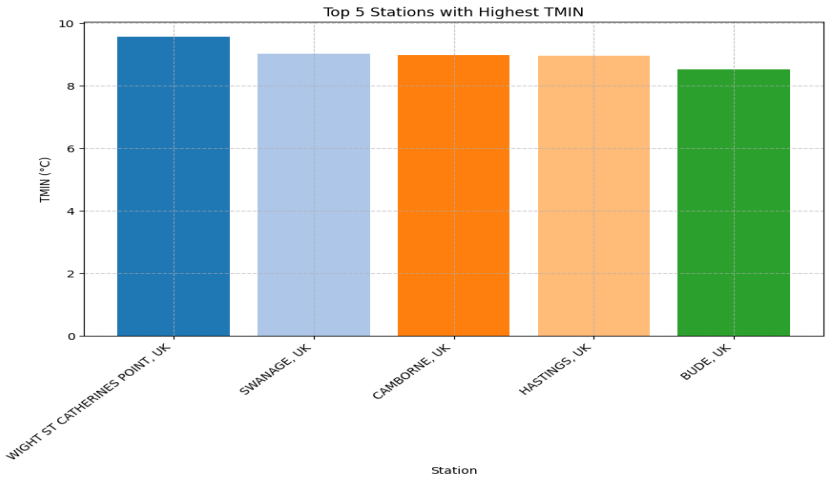


Figure 3:

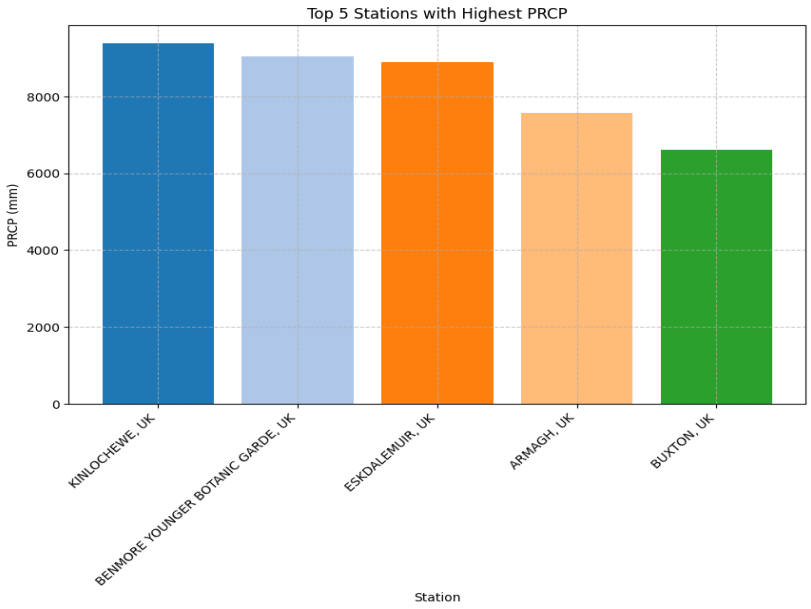


Figure 4:

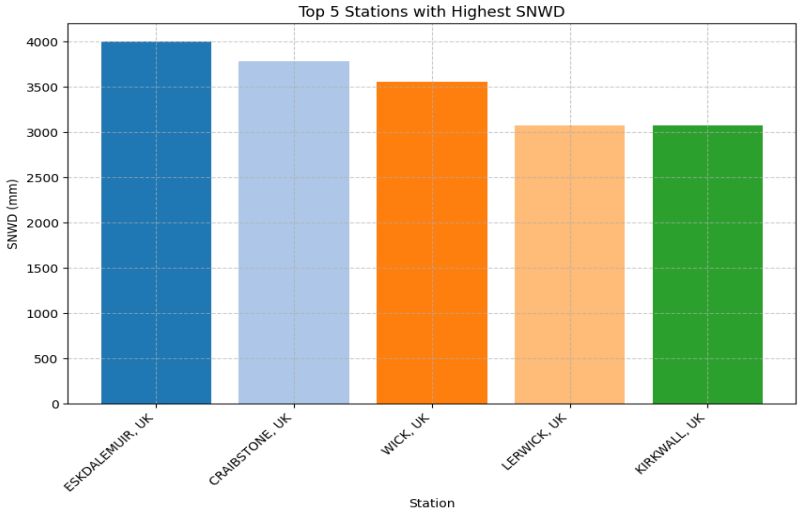


Figure 5:

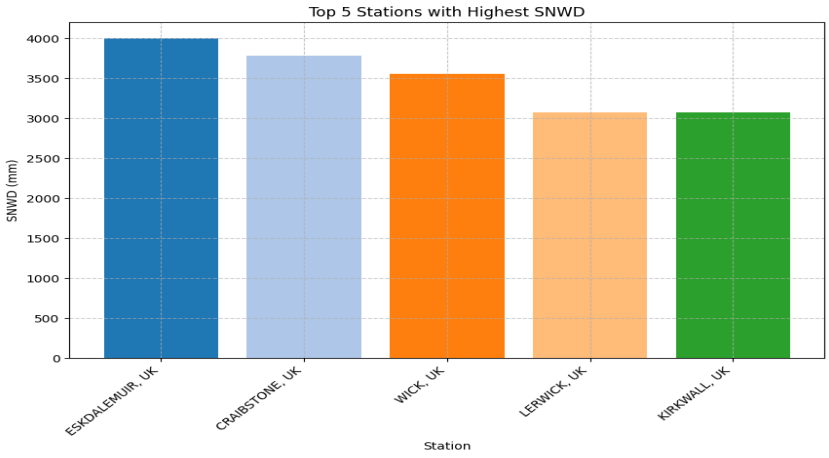
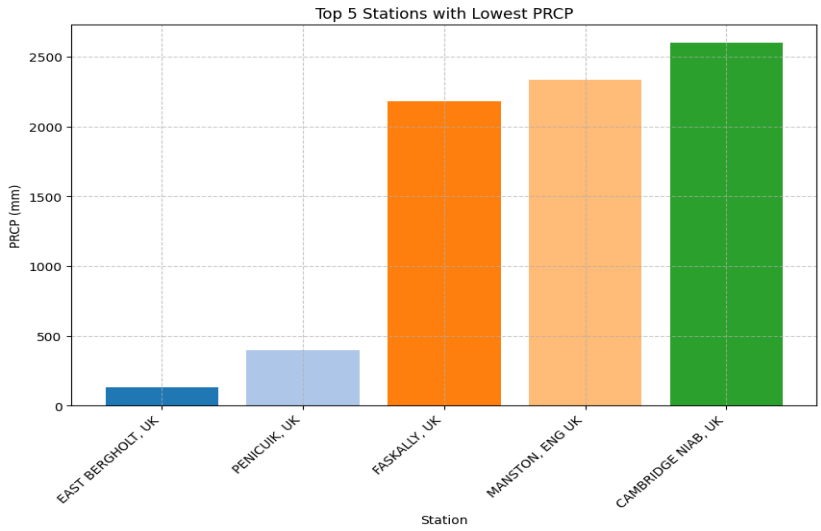


Figure 6:



6.2.3 CONCLUSION

The analysis effectively identifies and compares weather extremes across UK regions, providing insights into local climate patterns crucial for scientific research and practical applications such as climate research, agriculture, urban planning, and extreme weather monitoring. The findings align with the project's objectives, offering valuable information for disaster preparedness and management, particularly in regions prone to severe weather events like flooding or snowstorms. This approach ensures that the study's conclusions are both scientifically sound and practically relevant.

6.3 SUMMARY STATISTICS FOR WEATHER DATA

6.3.1 METRICS AND THEIR SIGNIFICANCE

For this analysis, I used several key statistical metrics to understand the central tendencies and dispersion of different weather parameters across daily, monthly, and yearly datasets. The metrics computed include:

- **Count:** The number of non-missing observations.
- **Mean:** The average value, giving a sense of the typical weather condition.
- **Standard Deviation (std):** The variability or spread of the data, indicating how much the values deviate from the mean.
- **Minimum (min) and Maximum (max) Values:** The extreme values recorded in the dataset.
- **25th Percentile (25%), Median (50%), and 75th Percentile (75%):** These metrics provide insights into the distribution of data, helping to understand the spread and skewness.

These metrics were calculated for various weather parameters, including maximum and minimum temperatures, average temperatures, precipitation, and snow depth.

6.3.2 RESULTS INTERPRETATION

The analysis of daily, monthly, and yearly weather data reveals several key insights:

- **Daily Data Summary:**
    - The average daily maximum temperature (TMAX) is 13.81°C, with a wide range from -9.3°C to 40.2°C. The average minimum temperature (TMIN) is 6.55°C.
    - Daily precipitation averages 2.75 mm, with a peak value of 140 mm, while snow depth (SNWD) averages 1.15 mm, reaching up to 279 mm on extreme days.
- **Monthly Data Summary:**
    - The mean extreme monthly precipitation (EMXP) is 18.11 mm, showing high variability with a standard deviation of 10.94 mm.
    - The average maximum temperature (EMXT) for the period is 19.54°C, with extreme highs reaching 40.2°C.
    - Additional metrics like snow depth days (DSND), high-temperature days (DX90), and heating degree days (HDSD) provide further insights into monthly climate conditions.
- **Yearly Data Summary:**
    - Yearly data mirrors monthly trends, with maximum yearly temperatures (TMAX) averaging 13.93°C and extremes between -20.9°C and 40.2°C.
    - Precipitation and temperature vary significantly across years, reflecting broader climatic trends.

6.3.3 PRESENTATION OF RESULTS

- **Tables:** The summary statistics were presented in tabular format (refer to Table 1 to Table 3), which is effective for displaying a comprehensive overview of the dataset. Tables allow for a clear and concise presentation of key metrics, making it easy to compare different weather parameters across daily, monthly, and yearly data.

Table 1:

Summary Statistics for Daily Data:

	TMAX	TAVG	TMIN	PRCP
count	134936.000000	134936.000000	134936.000000	134936.000000
mean	13.813249	9.942567	6.551323	2.747755
std	5.390773	2.384579	4.729702	5.277994
min	-9.300000	-8.100000	-23.000000	0.000000
25%	10.200000	9.942567	3.600000	0.000000
50%	13.813249	9.942567	6.551323	0.600000
75%	17.100000	9.942567	9.700000	2.747755
max	40.200000	29.800000	24.500000	140.000000

	SNWD
count	134936.000000
mean	1.147711
std	3.630856
min	0.000000
25%	1.147711
50%	1.147711
75%	1.147711
max	279.000000

Table 2:

Summary Statistics for Monthly Data:

	EMXP	EMXT	DSND	PRCP	DX90
count	4243.000000	4243.000000	4243.00000	4243.000000	4243.000000
mean	18.113306	19.542802	0.21673	82.697470	0.062535
std	10.938510	5.855839	0.37335	58.830828	0.370700
min	0.000000	6.700000	0.00000	0.000000	0.000000
25%	10.900000	14.700000	0.21673	41.800000	0.000000
50%	16.800000	19.542802	0.21673	74.800000	0.000000
75%	22.200000	23.300000	0.21673	107.700000	0.000000
max	118.800000	40.200000	13.00000	795.000000	6.000000

	DP10	HDSD	HTDD	DX70	DP01
count	4243.000000	4243.000000	4243.000000	4243.000000	4243.000000
mean	8.477106	1358.331772	241.310456	3.514452	15.718695
std	4.691576	858.544008	110.328604	5.812178	5.924943
min	0.000000	8.800000	0.800000	0.000000	0.000000
25%	5.000000	662.550000	163.200000	0.000000	12.000000
50%	8.477106	1358.331772	241.310456	0.000000	15.718695
75%	11.000000	1865.350000	325.700000	3.514452	20.000000
max	26.000000	4118.100000	598.700000	31.000000	31.000000

	CDSD	EMNT	DT32	DT00	DX32
count	4243.000000	4243.000000	4243.000000	4243.000000	4243.000000
mean	12.815562	0.604215	3.285714	0.000585	0.048638
std	19.223630	4.380060	4.196133	0.030703	0.334447
min	0.000000	-20.900000	0.000000	0.000000	0.000000
25%	0.000000	-2.400000	0.000000	0.000000	0.000000
50%	12.815562	0.604215	3.000000	0.000000	0.000000
75%	12.815562	3.500000	4.000000	0.000000	0.000000
max	137.900000	14.600000	28.000000	2.000000	6.000000

	CLDD	TMAX	EMSD	TAVG	TMIN
count	4243.000000	4243.000000	4243.000000	4243.000000	4243.000000
mean	2.596505	13.932490	6.026616	10.161465	6.491833
std	6.998268	4.642611	8.647255	3.999102	3.744026
min	0.000000	2.100000	0.000000	-1.000000	-4.100000
25%	0.000000	10.300000	6.026616	7.100000	3.700000
50%	0.000000	13.932490	6.026616	10.161465	6.491833
75%	2.596505	17.400000	6.026616	13.000000	9.300000
max	72.000000	27.300000	240.000000	20.500000	16.600000

Table 3:

Summary Statistics for Yearly Data:

	EMXP	EMXT	PRCP	DX90	DP10
count	4243.000000	4243.000000	4243.000000	4243.000000	4243.000000
mean	18.113306	19.542802	82.697470	0.062535	8.477106
std	10.938510	5.855839	58.830828	0.370700	4.691576
min	0.000000	6.700000	0.000000	0.000000	0.000000
25%	10.900000	14.700000	41.800000	0.000000	5.000000
50%	16.800000	19.542802	74.800000	0.000000	8.477106
75%	22.200000	23.300000	107.700000	0.000000	11.000000
max	118.800000	40.200000	795.000000	6.000000	26.000000

	DX70	EMNT	DT32	DX32	TMAX
count	4243.000000	4243.000000	4243.000000	4243.000000	4243.000000
mean	3.514452	0.604215	3.285714	0.048638	13.932490
std	5.812178	4.380060	4.196133	0.334447	4.642611
min	0.000000	-20.900000	0.000000	0.000000	2.100000
25%	0.000000	-2.400000	0.000000	0.000000	10.300000
50%	0.000000	0.604215	3.000000	0.000000	13.932490
75%	3.514452	3.500000	4.000000	0.000000	17.400000
max	31.000000	14.600000	28.000000	6.000000	27.300000

	EMSD	TAVG	TMIN
count	4243.000000	4243.000000	4243.000000
mean	6.026616	10.161465	6.491833
std	8.647255	3.999102	3.744026
min	0.000000	-1.000000	-4.100000
25%	6.026616	7.100000	3.700000
50%	6.026616	10.161465	6.491833
75%	6.026616	13.000000	9.300000
max	240.000000	20.500000	16.600000

6.3.4 CONCLUSION

The summary statistics offer essential insights into weather patterns, crucial for climate research, urban planning, and agriculture. The analysis of extreme values is valuable for assessing the impact of severe weather on infrastructure and communities. This aligns with the project objectives, providing a comprehensive overview that aids decision-making in weather-related fields. The analysis of daily, monthly, and yearly data highlights weather variability and trends, supporting real-world applications.

6.4 HISTOGRAMS AND STATISTICAL ANALYSIS OF MONTHLY DATA

6.4.1 METRICS AND THEIR SIGNIFICANCE:

This analysis uses histograms and statistical metrics to explore weather patterns in the monthly dataset. Key metrics include:

- **Mean:** Indicates the central tendency.
- **Standard Deviation (STD):** Reflects variability around the mean.
- **Skewness:** Shows the asymmetry of data distribution, with values close to zero indicating symmetry.
- **Kurtosis:** Measures the "tailedness" of the data distribution, where higher values suggest the presence of outliers.
- **Sample Size:** Represents the number of observations, indicating the reliability of the metrics.

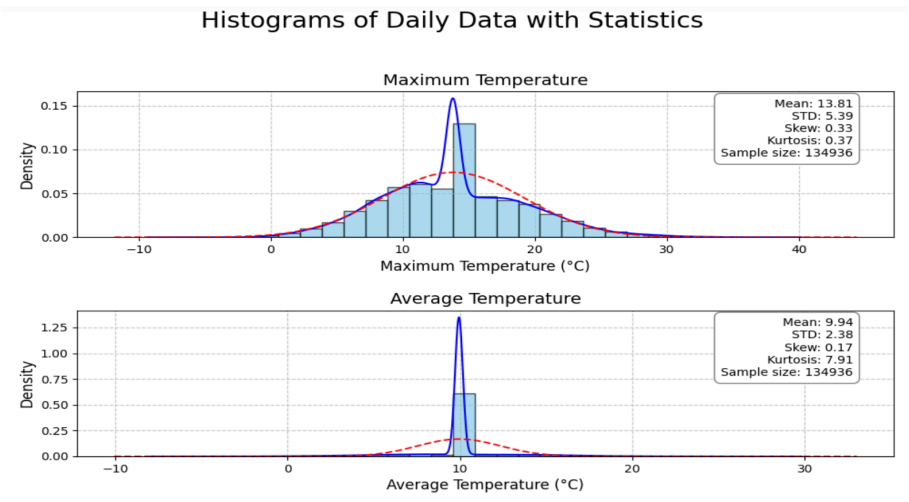
6.4.2 RESULTS INTERPRETATION:

- **Extreme Maximum Precipitation:** Positively skewed with a mean of 18.11 mm, showing variability in extreme events.
- **Extreme Maximum Temperature:** Slight positive skewness with a broad distribution around 19.54°C.
- **Precipitation:** Positively skewed, with variability indicated by a standard deviation reflecting differing rainfall patterns across months.
- **Days with Maximum Temperature > 90°F:** Mostly zero, indicating few extreme heat days.
- **Cooling Degree Days:** Highly skewed, indicating few months requiring significant cooling.
- **Extreme Minimum Temperature:** Negatively skewed, with extreme cold temperatures more frequent.
- **Extreme Maximum Snow Depth:** Positively skewed, showing occasional heavy snowfall.

6.4.3 PRESENTATION OF RESULTS:

- **Histogram Plots:** Provide a visual distribution of weather parameters, with KDE curves offering a smoothed data estimate (refer figure 1 to figure 3).
- **Statistical Annotations:** Included in each plot to easily interpret key data characteristics.

Figure 1:



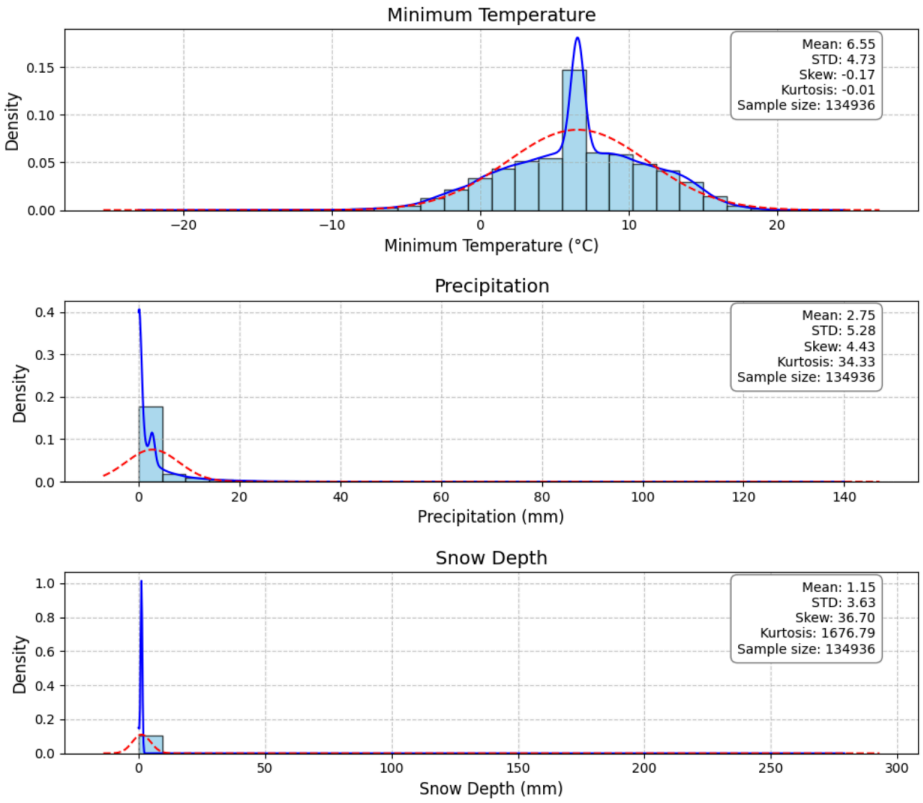
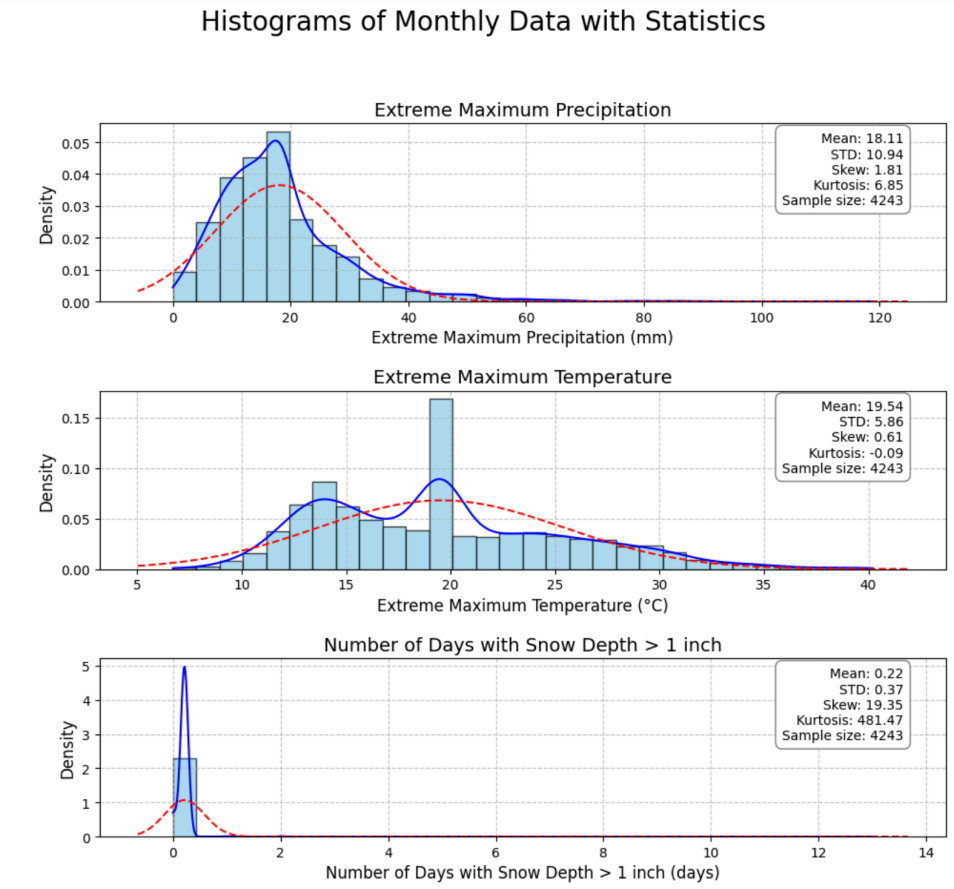
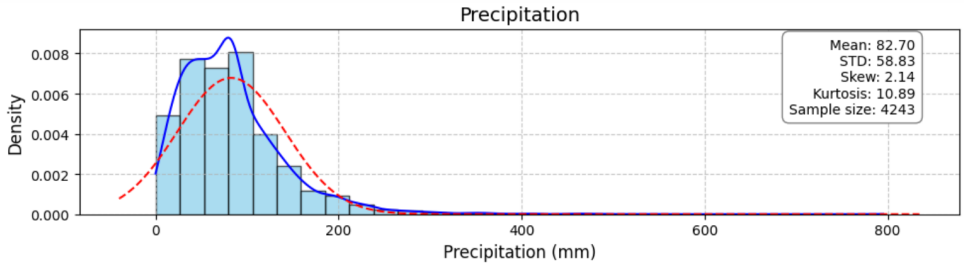
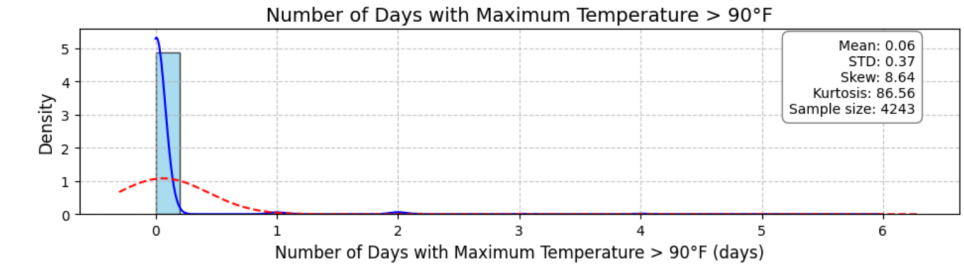
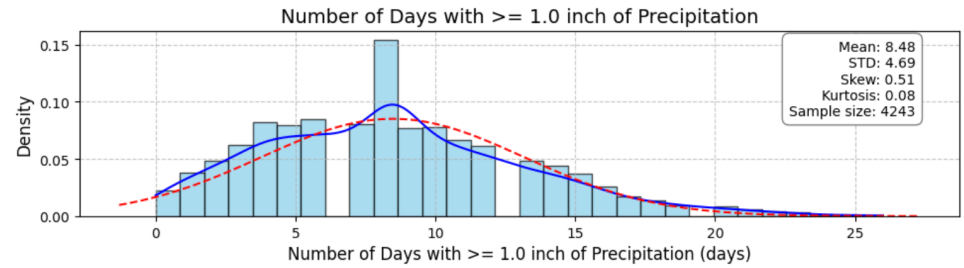

Minimum Temperature
Mean: 6.55
STD: 4.73
Skew: -0.17
Kurtosis: -0.01
Sample size: 134936
Density
Minimum Temperature (°C)
Precipitation
Mean: 2.75
STD: 5.28
Skew: 4.43
Kurtosis: 34.33
Sample size: 134936
Density
Precipitation (mm)
Snow Depth
Mean: 1.15
STD: 3.63
Skew: 36.70
Kurtosis: 1676.79
Sample size: 134936
Density
Snow Depth (mm)

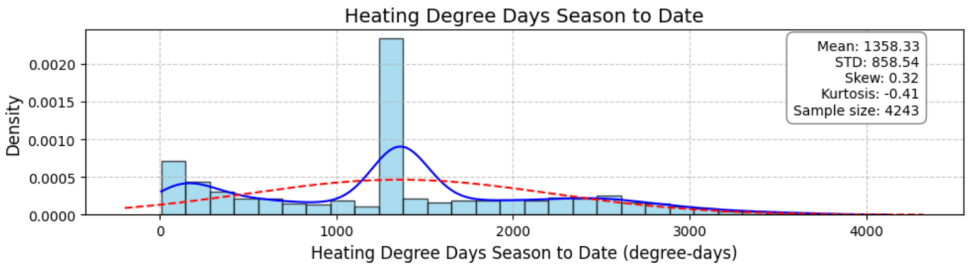
Figure 2:

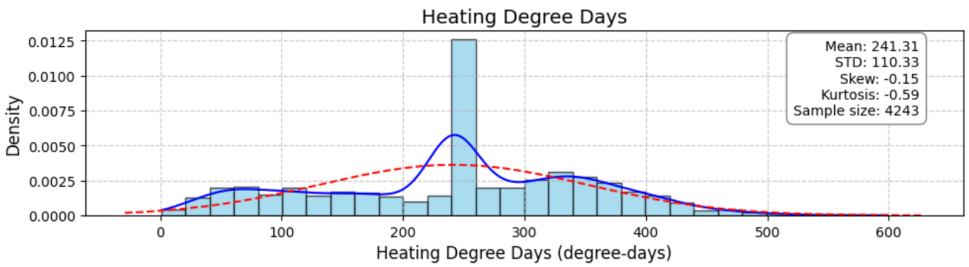

Histograms of Monthly Data with Statistics
Extreme Maximum Precipitation
Mean: 18.11
STD: 10.94
Skew: 1.81
Kurtosis: 6.85
Sample size: 4243
Density
Extreme Maximum Precipitation (mm)
Extreme Maximum Temperature
Mean: 19.54
STD: 5.86
Skew: 0.61
Kurtosis: -0.09
Sample size: 4243
Density
Extreme Maximum Temperature (°C)
Number of Days with Snow Depth > 1 inch
Mean: 0.22
STD: 0.37
Skew: 19.35
Kurtosis: 481.47
Sample size: 4243
Density
Number of Days with Snow Depth > 1 inch (days)

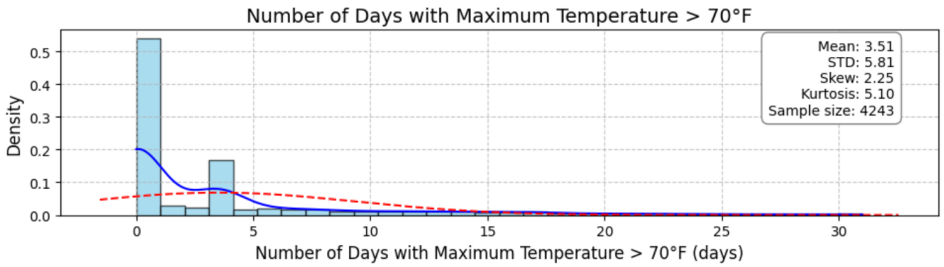

Precipitation
Mean: 82.70
STD: 58.83
Skew: 2.14
Kurtosis: 10.89
Sample size: 4243
Density
Precipitation (mm)

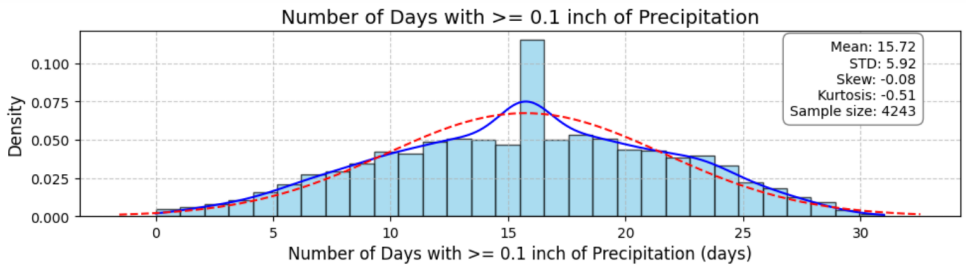

Number of Days with Maximum Temperature > 90°F
Mean: 0.06
STD: 0.37
Skew: 8.64
Kurtosis: 86.56
Sample size: 4243
Density
Number of Days with Maximum Temperature > 90°F (days)

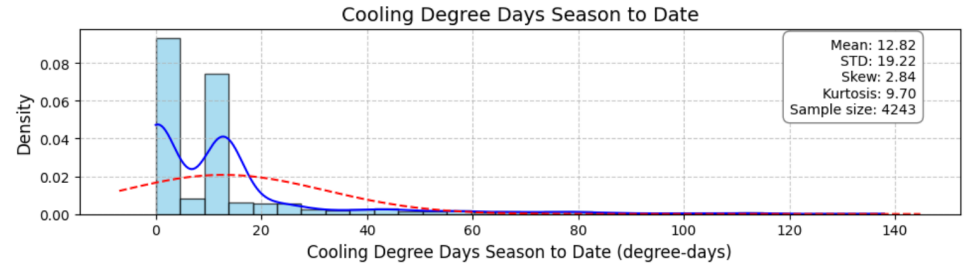

Number of Days with >= 1.0 inch of Precipitation
Mean: 8.48
STD: 4.69
Skew: 0.51
Kurtosis: 0.08
Sample size: 4243
Density
Number of Days with >= 1.0 inch of Precipitation (days)

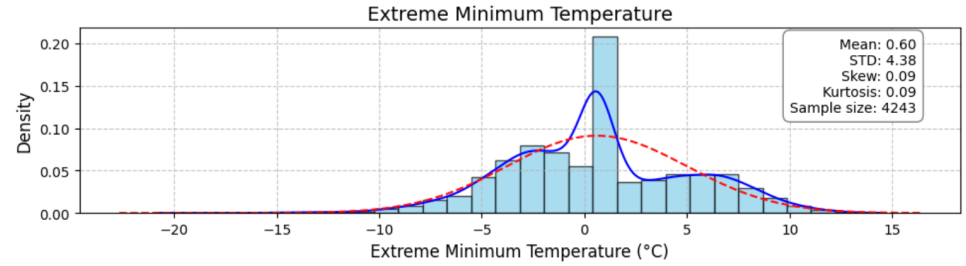

Heating Degree Days Season to Date
Mean: 1358.33
STD: 858.54
Skew: 0.32
Kurtosis: -0.41
Sample size: 4243
Density
Heating Degree Days Season to Date (degree-days)

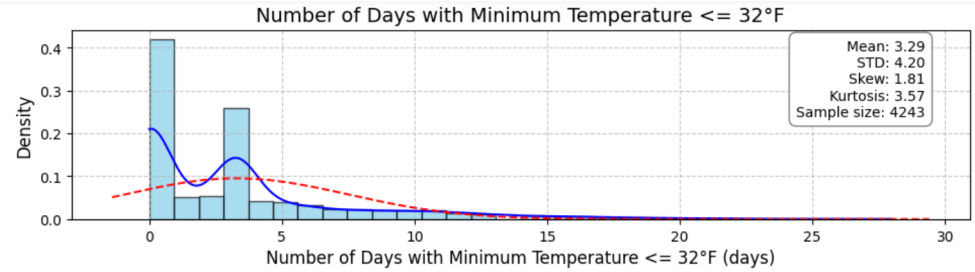

Heating Degree Days
Mean: 241.31
STD: 110.33
Skew: -0.15
Kurtosis: -0.59
Sample size: 4243
Density
Heating Degree Days (degree-days)

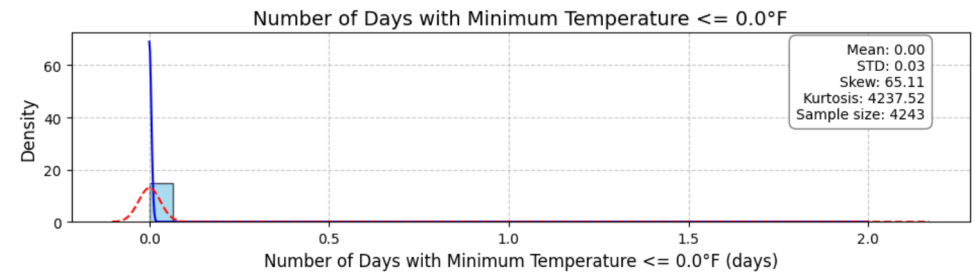

Number of Days with Maximum Temperature > 70°F
Mean: 3.51
STD: 5.81
Skew: 2.25
Kurtosis: 5.10
Sample size: 4243
Density
Number of Days with Maximum Temperature > 70°F (days)

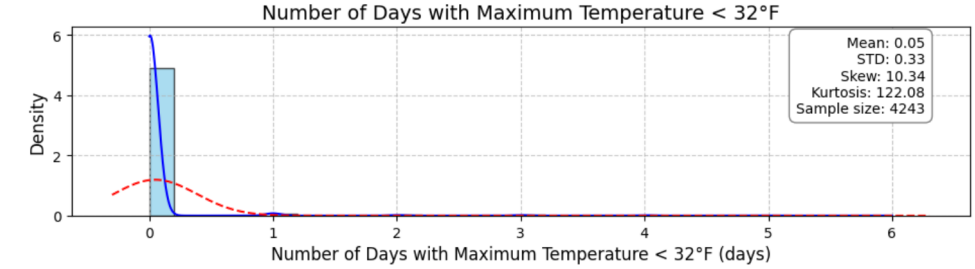

Number of Days with >= 0.1 inch of Precipitation
Mean: 15.72
STD: 5.92
Skew: -0.08
Kurtosis: -0.51
Sample size: 4243
Density
Number of Days with >= 0.1 inch of Precipitation (days)

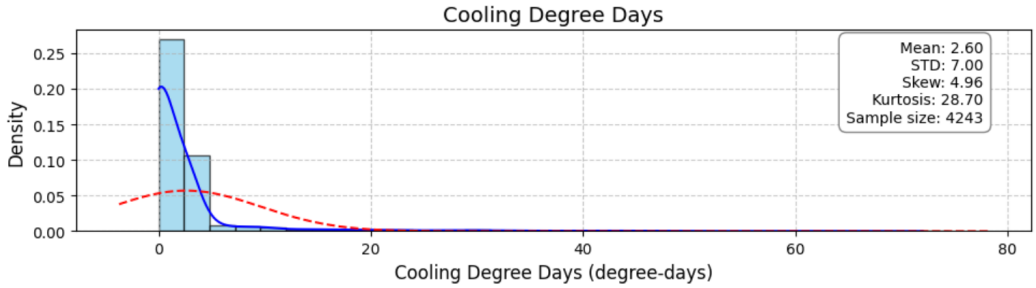

Cooling Degree Days Season to Date
Mean: 12.82
STD: 19.22
Skew: 2.84
Kurtosis: 9.70
Sample size: 4243
Density
Cooling Degree Days Season to Date (degree-days)

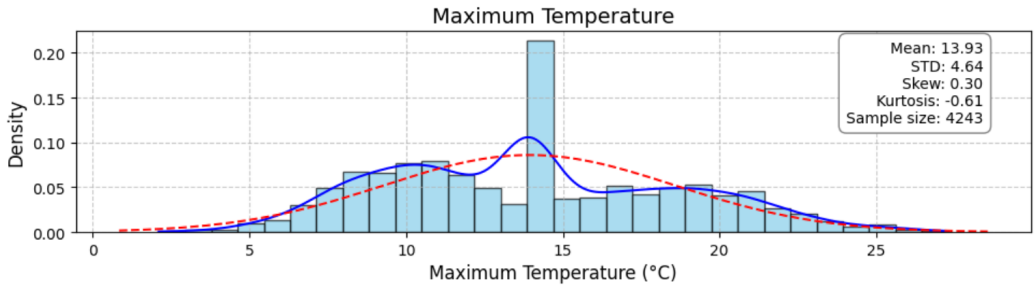

Extreme Minimum Temperature
Mean: 0.60
STD: 4.38
Skew: 0.09
Kurtosis: 0.09
Sample size: 4243
Density
Extreme Minimum Temperature (°C)

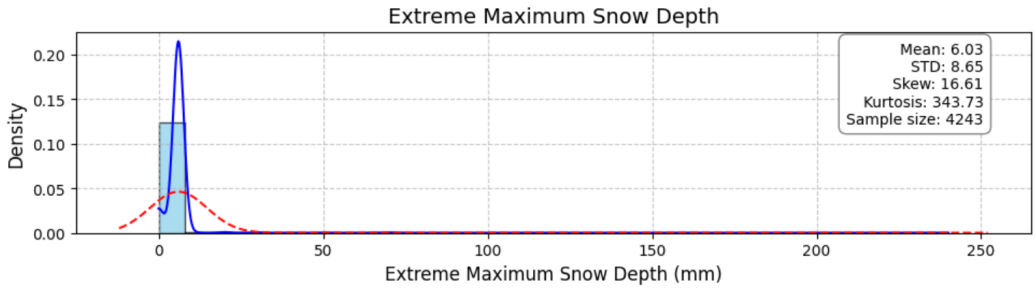

Number of Days with Minimum Temperature <= 32°F
Mean: 3.29
STD: 4.20
Skew: 1.81
Kurtosis: 3.57
Sample size: 4243
Density
Number of Days with Minimum Temperature <= 32°F (days)

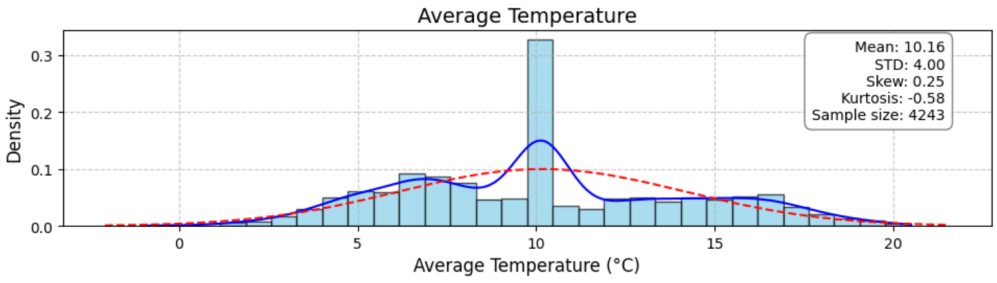

Number of Days with Minimum Temperature <= 0.0°F
Mean: 0.00
STD: 0.03
Skew: 65.11
Kurtosis: 4237.52
Sample size: 4243
Density
Number of Days with Minimum Temperature <= 0.0°F (days)


Number of Days with Maximum Temperature < 32°F
Mean: 0.05
STD: 0.33
Skew: 10.34
Kurtosis: 122.08
Sample size: 4243
Density
Number of Days with Maximum Temperature < 32°F (days)


Cooling Degree Days
Mean: 2.60
STD: 7.00
Skew: 4.96
Kurtosis: 28.70
Sample size: 4243
Density
Cooling Degree Days (degree-days)


Maximum Temperature
Mean: 13.93
STD: 4.64
Skew: 0.30
Kurtosis: -0.61
Sample size: 4243
Density
Maximum Temperature (°C)


Extreme Maximum Snow Depth
Mean: 6.03
STD: 8.65
Skew: 16.61
Kurtosis: 343.73
Sample size: 4243
Density
Extreme Maximum Snow Depth (mm)


Average Temperature
Mean: 10.16
STD: 4.00
Skew: 0.25
Kurtosis: -0.58
Sample size: 4243
Density
Average Temperature (°C)

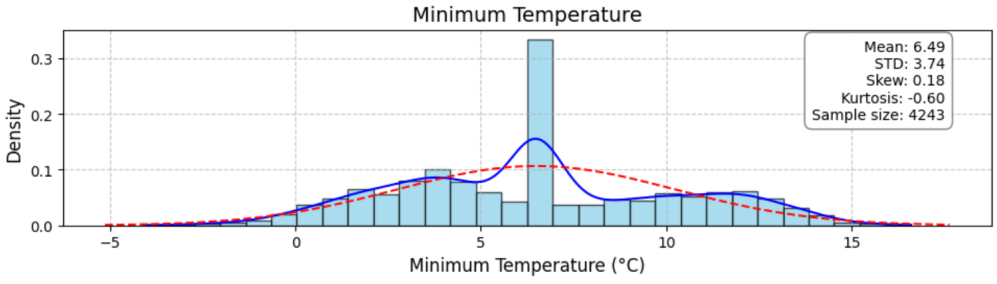
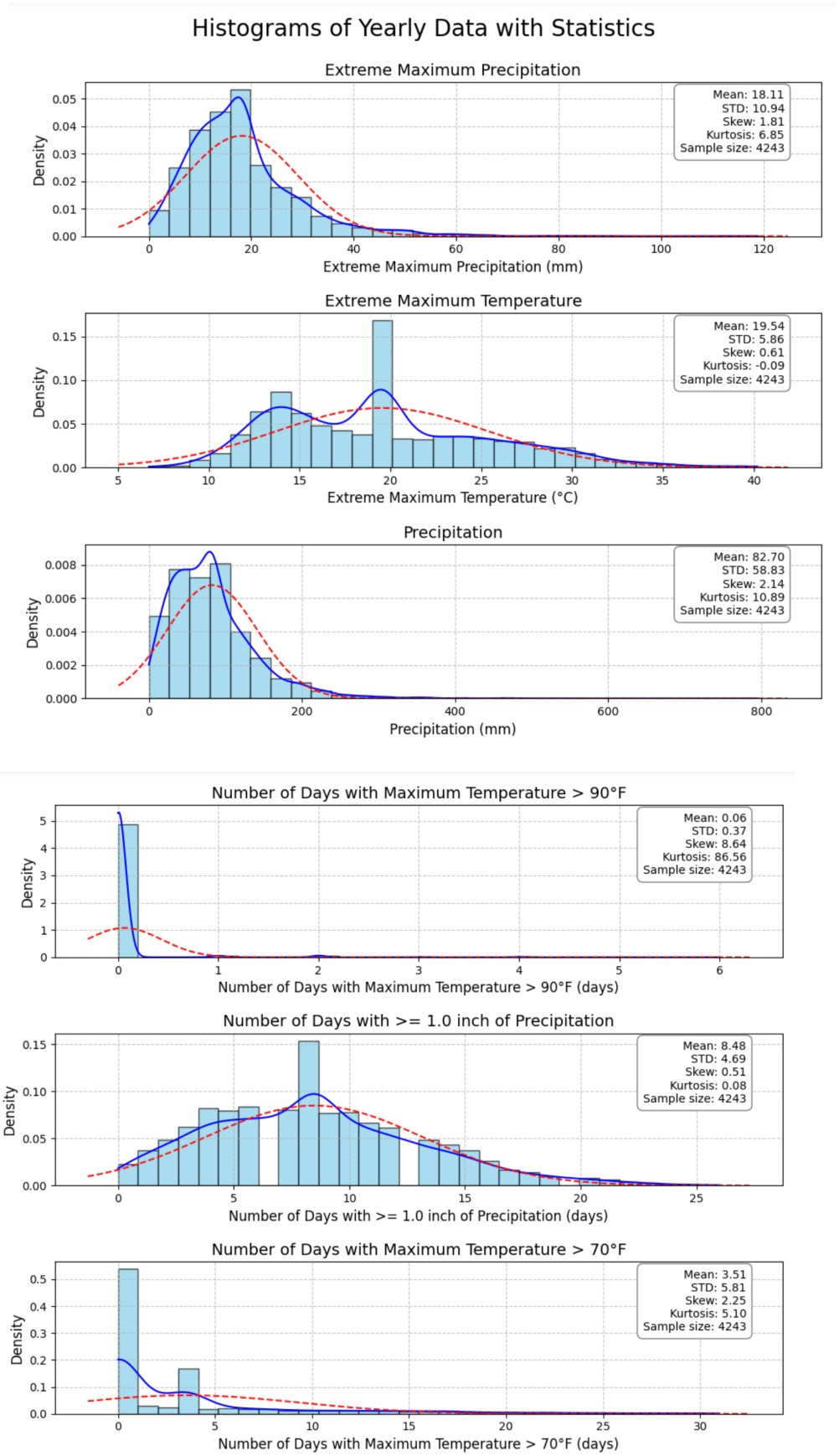
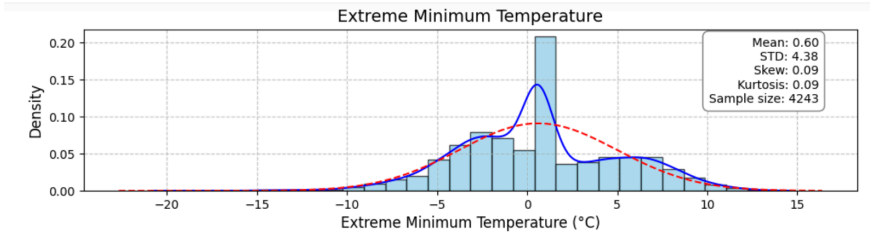
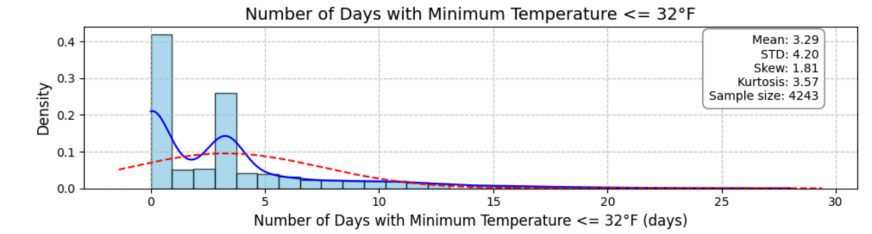
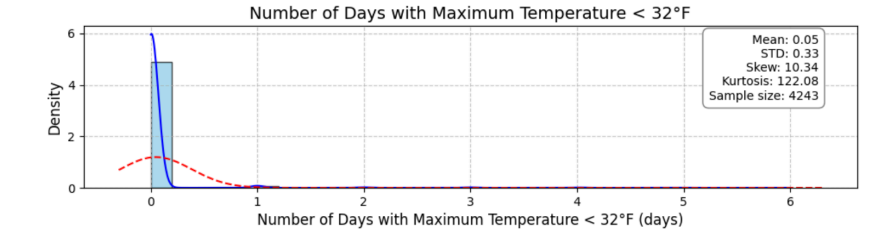

Minimum Temperature
Mean: 6.49
STD: 3.74
Skew: 0.18
Kurtosis: -0.60
Sample size: 4243
Density
Minimum Temperature (°C)

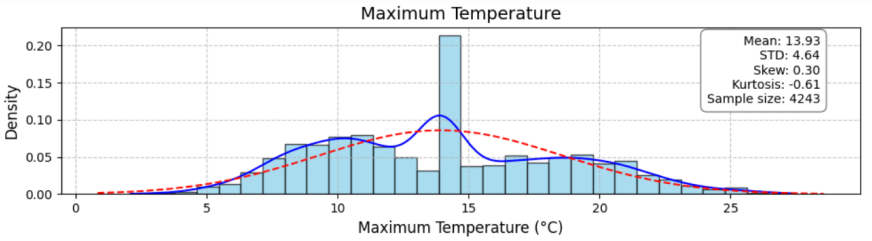
Figure 3:

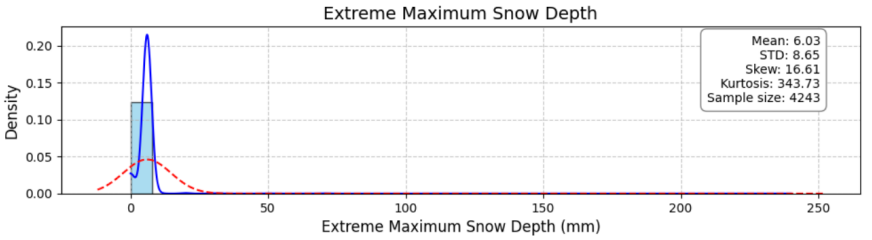

Histograms of Yearly Data with Statistics
Extreme Maximum Precipitation
Mean: 18.11
STD: 10.94
Skew: 1.81
Kurtosis: 6.85
Sample size: 4243
Density
Extreme Maximum Precipitation (mm)
Extreme Maximum Temperature
Mean: 19.54
STD: 5.86
Skew: 0.61
Kurtosis: -0.09
Sample size: 4243
Density
Extreme Maximum Temperature (°C)
Precipitation
Mean: 82.70
STD: 58.83
Skew: 2.14
Kurtosis: 10.89
Sample size: 4243
Density
Precipitation (mm)
Number of Days with Maximum Temperature > 90°F
Mean: 0.06
STD: 0.37
Skew: 8.64
Kurtosis: 86.56
Sample size: 4243
Density
Number of Days with Maximum Temperature > 90°F (days)
Number of Days with >= 1.0 inch of Precipitation
Mean: 8.48
STD: 4.69
Skew: 0.51
Kurtosis: 0.08
Sample size: 4243
Density
Number of Days with >= 1.0 inch of Precipitation (days)
Number of Days with Maximum Temperature > 70°F
Mean: 3.51
STD: 5.81
Skew: 2.25
Kurtosis: 5.10
Sample size: 4243
Density
Number of Days with Maximum Temperature > 70°F (days)

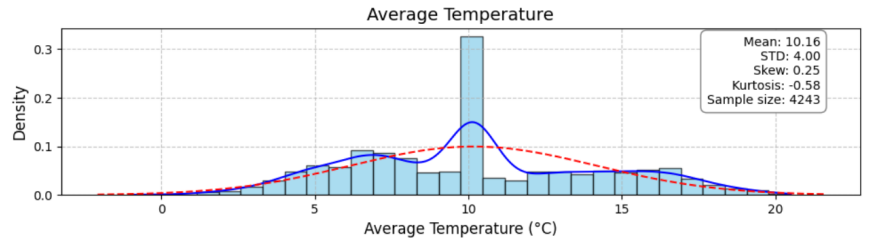

Extreme Minimum Temperature
Mean: 0.60
STD: 4.38
Skew: 0.09
Kurtosis: 0.09
Sample size: 4243
Density
Extreme Minimum Temperature (°C)

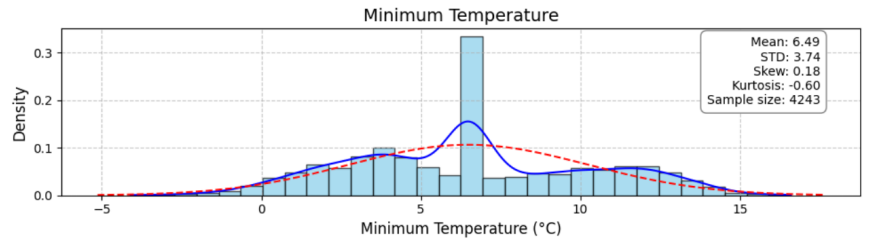

Number of Days with Minimum Temperature <= 32°F
Mean: 3.29
STD: 4.20
Skew: 1.81
Kurtosis: 3.57
Sample size: 4243
Density
Number of Days with Minimum Temperature <= 32°F (days)


Number of Days with Maximum Temperature < 32°F
Mean: 0.05
STD: 0.33
Skew: 10.34
Kurtosis: 122.08
Sample size: 4243
Density
Number of Days with Maximum Temperature < 32°F (days)


Maximum Temperature
Mean: 13.93
STD: 4.64
Skew: 0.30
Kurtosis: -0.61
Sample size: 4243
Density
Maximum Temperature (°C)


Extreme Maximum Snow Depth
Mean: 6.03
STD: 8.65
Skew: 16.61
Kurtosis: 343.73
Sample size: 4243
Density
Extreme Maximum Snow Depth (mm)


Average Temperature
Mean: 10.16
STD: 4.00
Skew: 0.25
Kurtosis: -0.58
Sample size: 4243
Density
Average Temperature (°C)


Minimum Temperature
Mean: 6.49
STD: 3.74
Skew: 0.18
Kurtosis: -0.60
Sample size: 4243
Density
Minimum Temperature (°C)

6.4.4 CONCLUSION

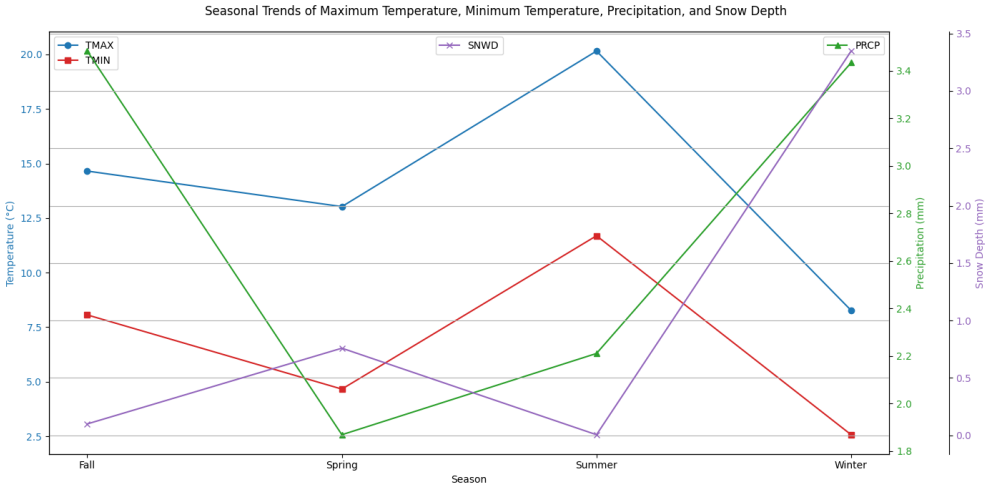
The histograms and statistical analysis offer detailed insights into the variability and distribution of key weather parameters. This information is vital for understanding climate patterns, risk assessment, and energy planning, aiding in informed decision-making in climate-related applications.

6.5 WEATHER DATA ANALYSIS

The analysis of the weather data was conducted using various statistical and visualization techniques to uncover trends and variability across different time frames (seasonal, monthly, and yearly). The results are summarized below:

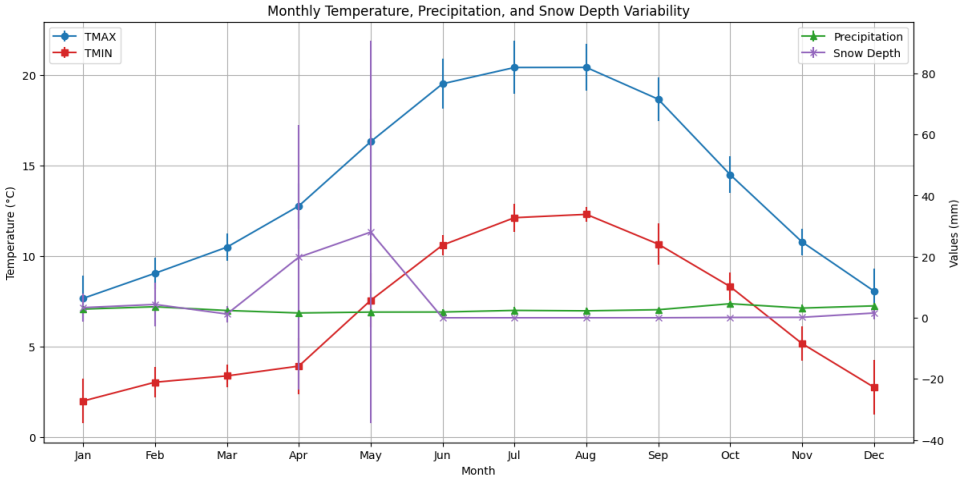
6.5.1 SEASONAL TRENDS

Maximum temperatures peak in summer and drop in winter, while precipitation shows an inverse trend, peaking in winter. Snow depth is highest during winter, corresponding to colder temperatures. These findings are crucial for agricultural planning, disaster management, and climate forecasting (Refer Figure: 1).



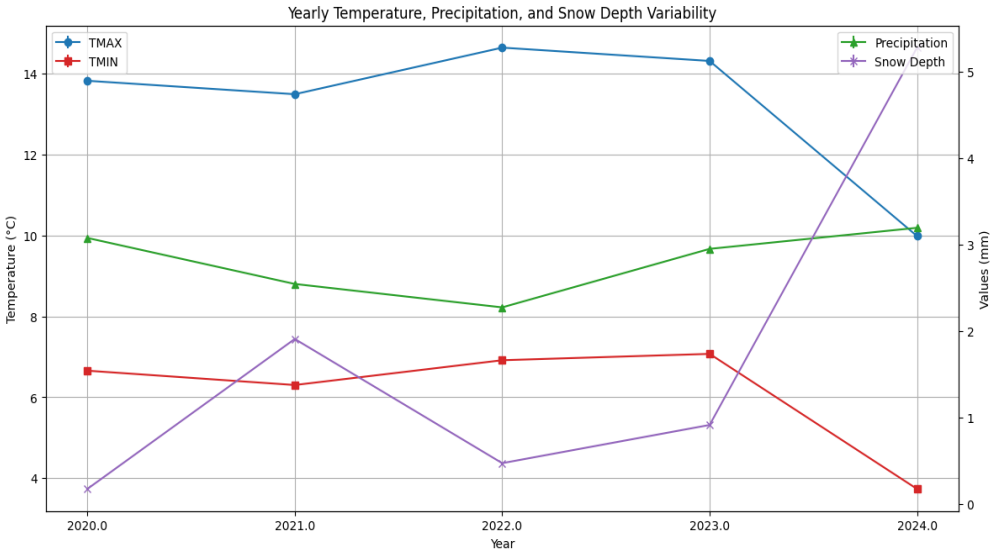
6.5.2 MONTHLY VARIABILITY

Temperatures steadily increase from January to July, peaking mid-year, and then decline towards December. Precipitation remains consistent with slight increases in spring, while snow depth varies significantly, particularly in spring and early summer. This variability is vital for infrastructure maintenance and energy consumption forecasting (Refer Figure: 2).



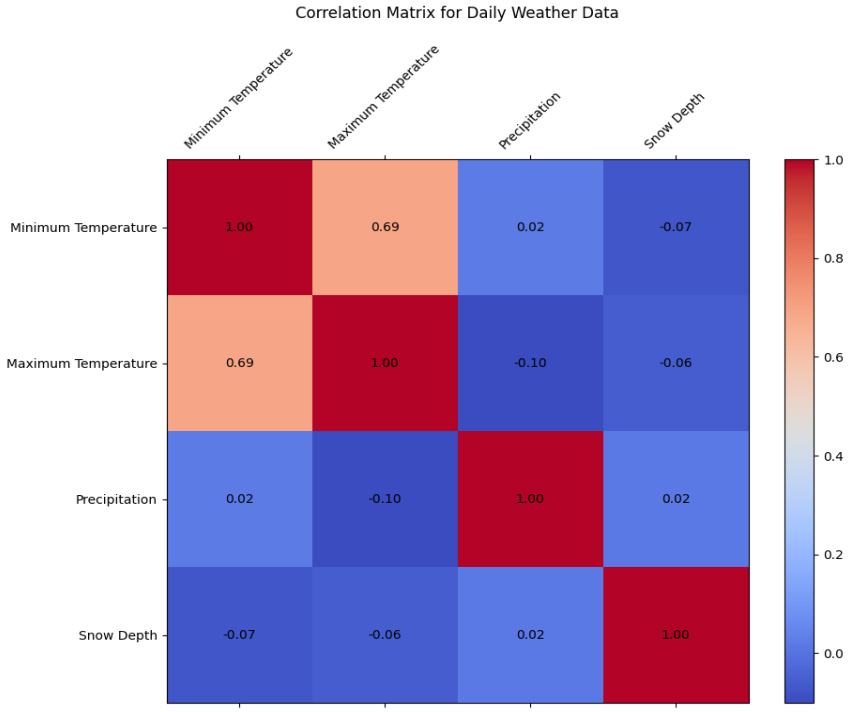
6.5.3 YEARLY VARIABILITY

There is a slight increase in maximum temperatures over the years, suggesting a warming trend, while snow depth generally declines, possibly indicating climate change. Precipitation shows fluctuations without a clear trend, highlighting the importance of long-term climate policy planning (Refer Figure: 3).

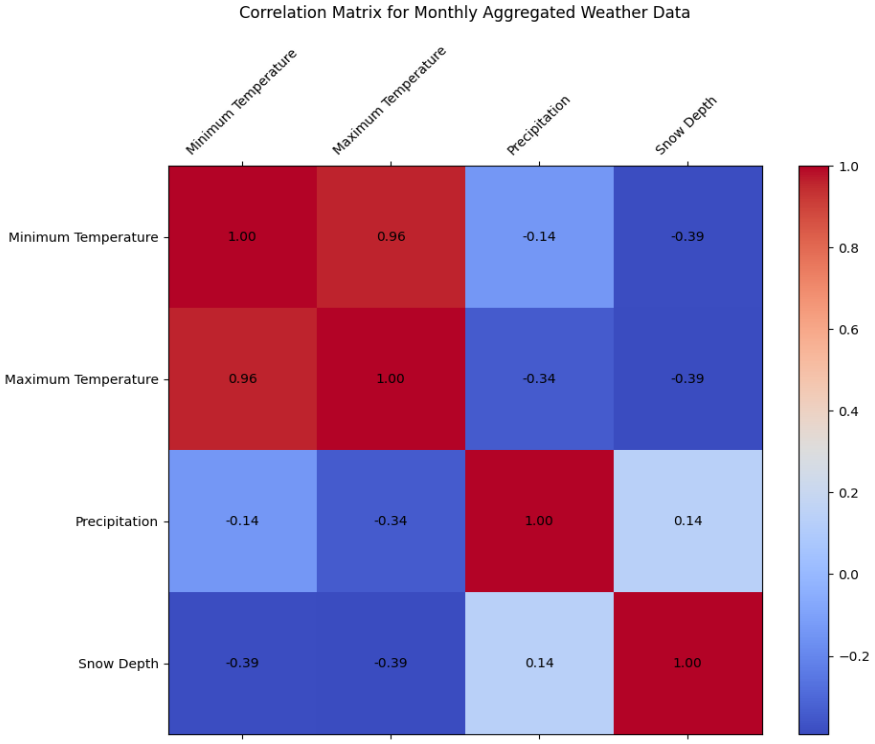


6.6 CORRELATION ANALYSIS:

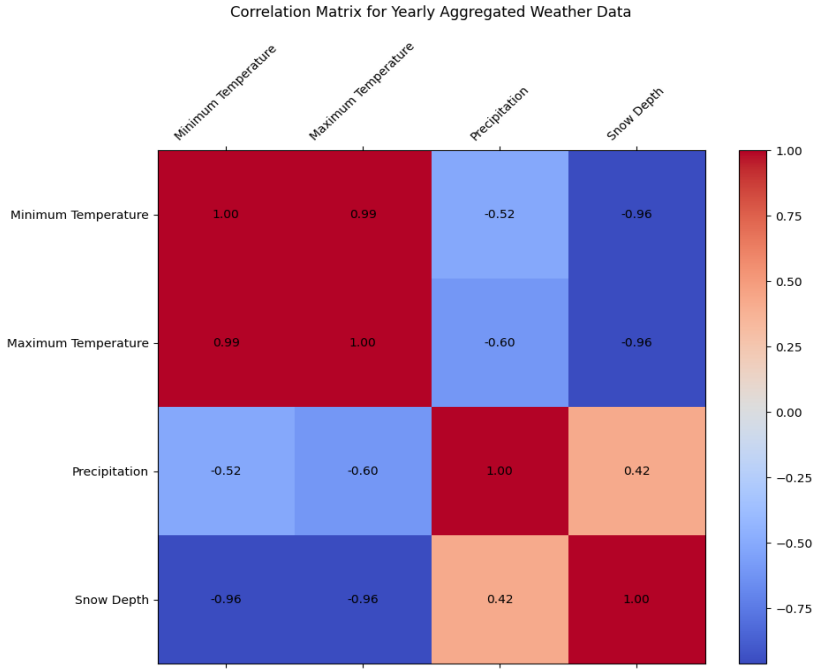
- **Daily Data:** A moderate positive correlation (0.69) between minimum and maximum temperatures indicates that higher minimum temperatures often accompany higher maximum temperatures. However, precipitation and snow depth show weak correlations with temperature, suggesting independent patterns.



- **Monthly and Yearly Correlations:** Strong correlations (up to 0.99) between temperature variables over longer time frames, with snow depth negatively correlated with temperature, as expected with higher temperatures reducing snow accumulation.



- **Climate Monitoring Implications:** These correlations highlight the importance of understanding interactions between weather elements, especially when assessing climate change impacts that may alter temperature and precipitation patterns.



6.6.1 CONCLUSION

The weather data analysis revealed key patterns in temperature, precipitation, and snow depth across daily, monthly, and yearly scales. It identified extreme events and their timing, highlighting periods of potential climate stress. Correlation analysis demonstrated the interconnectedness of weather variables, offering insights essential for regional climate resilience planning.

6.7 OUTLIER DETECTION IN WEATHER DATA SUMMARY

Identify and analyse outliers in daily weather data (TMAX, TMIN, PRCP, SNWD) using the Interquartile Range (IQR) method.

6.7.1 METHODOLOGY

- **IQR Method:** Outliers identified by calculating Q1, Q3, and the IQR. Points falling outside Q1 - 1.5 * IQR or Q3 + 1.5 * IQR are marked as outliers.
- **Visualization:** Plots highlight outliers, marked in red, against the original data (Refer Figure: 1 to Figure 4)

Figure 1:

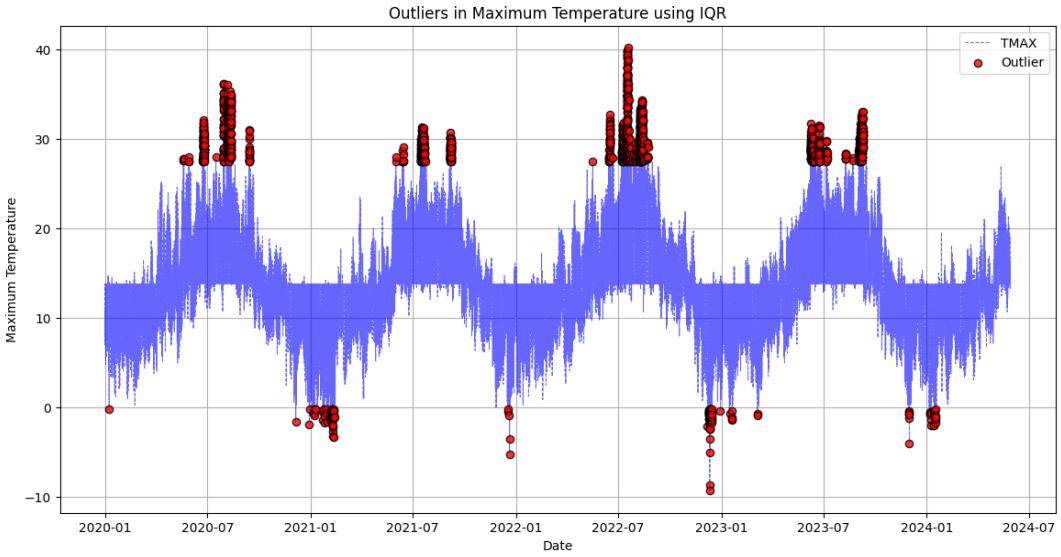


Figure 2 :

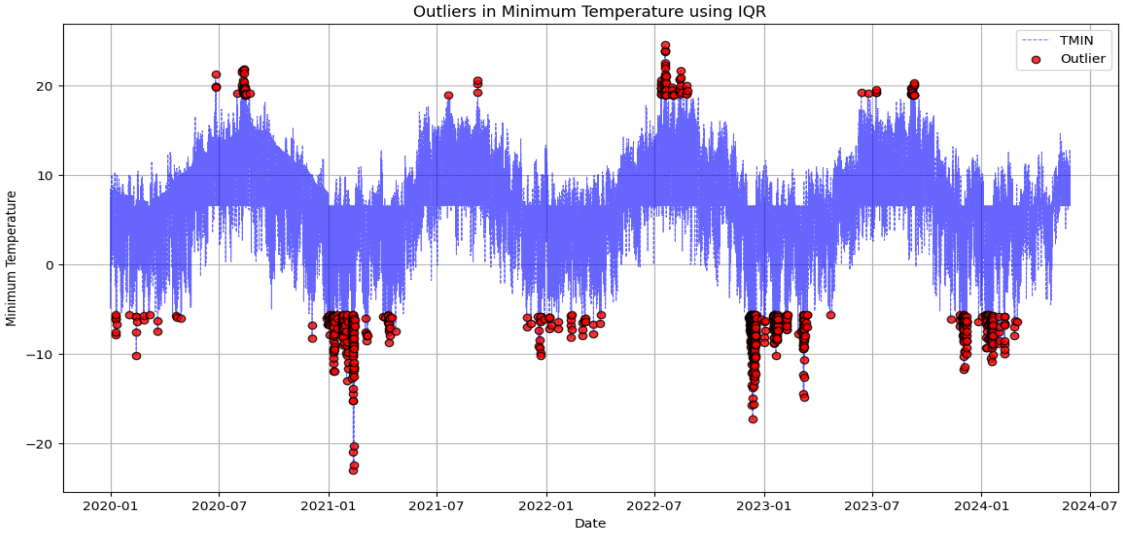


Figure 3 :

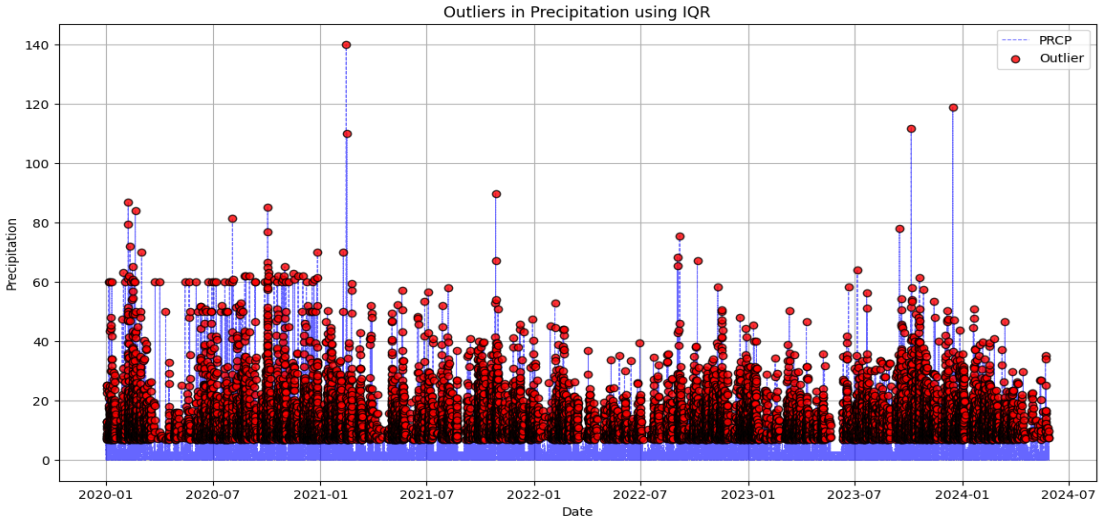
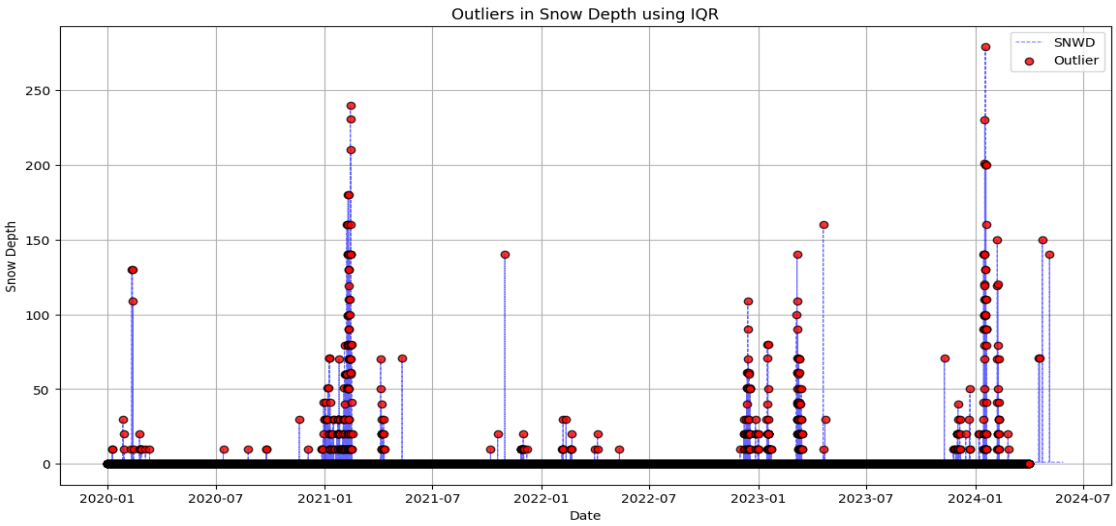


Figure 4:



6.7.2 SUMMARY STATISTICS

Count, mean, and standard deviation of outliers were calculated for each parameter.

6.7.3 RESULTS:

- TMAX: 1,819 outliers, mean of 26.65°C, range: -9.3°C to 40.2°C (refer figure 5).
- TMIN: 991 outliers, mean of -3.97°C, range: -23.0°C to 24.5°C (refer figure 6).
- PRCP: 16,066 outliers, mean of 14.01 mm, range: 6.9 mm to 140.0 mm (refer figure 7).
- SNWD: Significant outliers, particularly in winter, indicating extreme snowfall events(refer figure 8).

Figure 5: Column: TMAX Number of Outliers: 1819 Mean of Outliers: 26.65 Standard Deviation of Outliers: 9.75 Minimum Outlier Value: -9.30 Maximum Outlier Value: 40.20	Figure 6: Column: TMIN Number of Outliers: 991 Mean of Outliers: -3.97 Standard Deviation of Outliers: 9.32 Minimum Outlier Value: -23.00 Maximum Outlier Value: 24.50
Figure 7: Column: PRCP Number of Outliers: 16066 Mean of Outliers: 14.01 Standard Deviation of Outliers: 8.34 Minimum Outlier Value: 6.90 Maximum Outlier Value: 140.00	Figure 8: Column: SNWD Number of Outliers: 17365 Mean of Outliers: 1.15 Standard Deviation of Outliers: 10.12 Minimum Outlier Value: 0.00 Maximum Outlier Value: 279.00

6.7.4 CONCLUSIONS

Outliers quantify extreme weather events, providing insights into their nature and frequency. This analysis is valuable for climate studies, disaster preparedness, and resource management.

6.8 FORECASTING AND MODEL COMPARISON SUMMARY

6.8.1 PROPHET MODEL

- Objective: Forecast maximum temperature (TMAX).
- Evaluation Metrics (Refer Figure 1)

Figure 1:

```
Prophet RMSE: 111.64
Prophet MAE: 95.36
```

- **Conclusion:** The Prophet model, while effective in certain forecasting contexts, struggled with this dataset, evidenced by its higher RMSE and MAE values. The model's performance suggests that it may not be the best fit for capturing the complex, non-linear patterns inherent in this weather data.

6.8.2 ARIMA MODEL

- Objective: Forecast TMAX using a statistical approach.
- Evaluation Metrics (Refer Figure 2):

Figure 2:

```
ARIMA RMSE: 6.16846824037105
ARIMA MAE: 5.1416042678317275
```

- **Conclusion:** The ARIMA model provided a much-improved fit compared to Prophet, with significantly lower error metrics. Its ability to model linear relationships and time series dependencies contributed to its relative success, though it may still fall short in scenarios involving highly non-linear data.

6.8.3 LSTM MODEL

- **Objective:** Forecast TMAX using a Long Short-Term Memory neural network.
- Evaluation Metrics (Refer Figure 3):

Figure 3:

```
LSTM RMSE: 2.86
LSTM MAE: 2.15
```

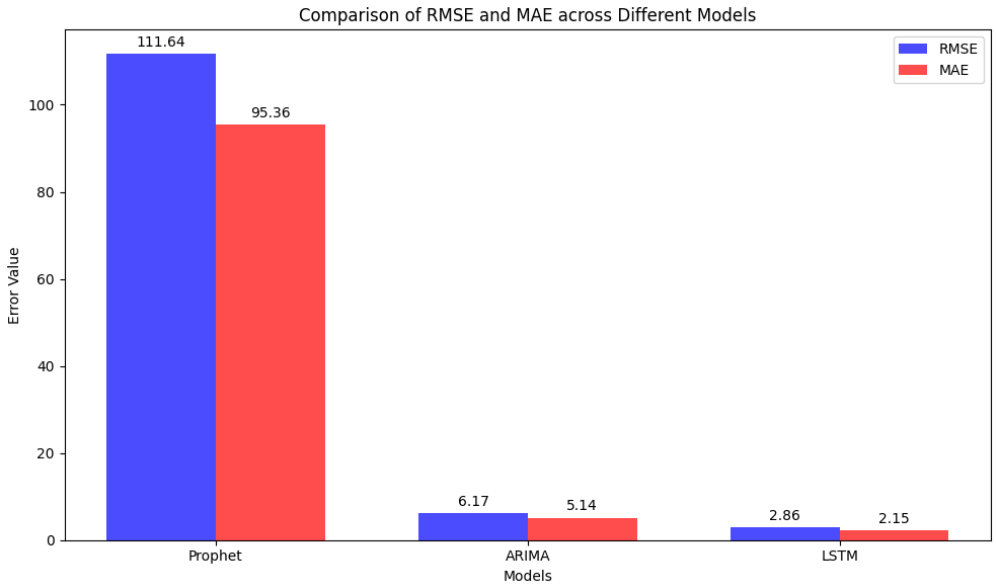
- **Conclusion:** The LSTM model outperformed both Prophet and ARIMA, demonstrating superior capability in capturing the temporal dependencies and non-linear dynamics of the weather data. Its architecture, designed to manage sequences and long-term dependencies, made it particularly effective for this forecasting task.

6.8.4 MODEL PERFORMANCE COMPARISON

The bar chart provides a visual comparison of the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) across three different models: Prophet, ARIMA, and LSTM, which were used to predict maximum temperatures (TMAX) (refer Figure 4)

- **Prophet:** Exhibited the highest error rates, suggesting it might be less suitable for datasets with complex patterns.
- **ARIMA:** Showed solid performance, particularly in capturing the linear aspects of the data.
- **LSTM:** Emerged as the most accurate, with the lowest RMSE and MAE, confirming its strength in dealing with complex, time-dependent data

Figure 4:



6.9 LSTM MODEL WITH TIME SERIES CROSS-VALIDATION

6.9.1 MODEL PERFORMANCE:

Figure 5:

```
LSTM Cross-Validation RMSE: 2.11
LSTM Cross-Validation MAE: 1.56
```

The LSTM model shows strong performance, with low RMSE and MAE values indicating accurate predictions. RMSE quantifies the model's average prediction error magnitude, while MAE measures the average absolute difference between predicted and actual values, highlighting the model's precision.

6.10 LSTM MODEL PERFORMANCE AND FORECASTING

6.10.1 MODEL PERFORMANCE

The LSTM model was trained and evaluated on daily weather data, with the model's effectiveness measured using Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The results for each parameter are summarized below :

	Parameter	RMSE	MAE
0	TMAX	2.868296	2.212443
1	TMIN	2.838441	2.166408
2	PRCP	5.307875	3.123499
3	SNWD	5.092840	0.998403
4	TMAX_ROLLING	1.497869	0.909283
5	TMIN_ROLLING	1.243949	0.671234
6	PRCP_ROLLING	1.314627	0.742082
7	SNWD_ROLLING	0.000882	0.000858

Figure 6:

6.10.2 INTERPRETATION

The model demonstrated strong predictive performance, particularly for temperature-related parameters, as evidenced by the relatively low RMSE and MAE values. The rolling averages (7-day) further improved prediction accuracy, especially for snow depth, where errors were minimized.

6.10.3 FORECASTING RESULTS

The LSTM model was used to forecast future values for several weather parameters, with predictions extending three years into the future. Key visualizations included the forecasted trends for maximum, average, and minimum temperatures, as well as precipitation, with corresponding 95% confidence intervals (refer figure 7 to figure 10)

Figure 7:

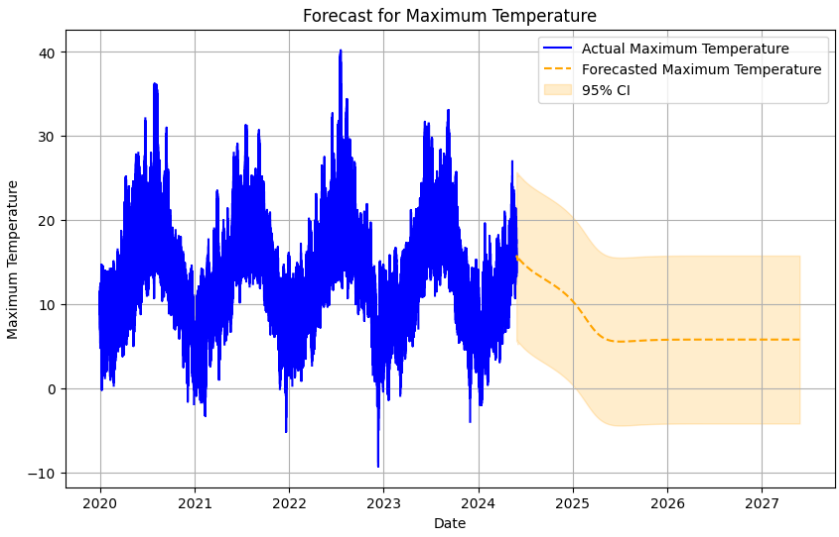


Figure 8:

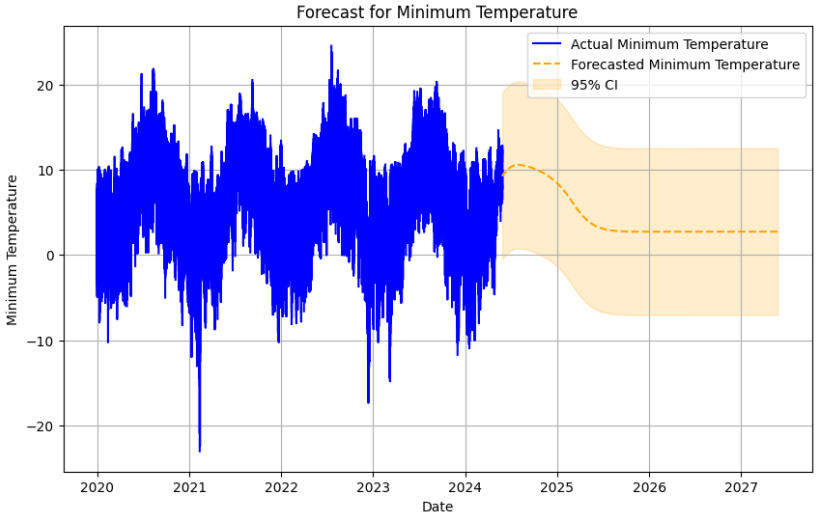


Figure 9:

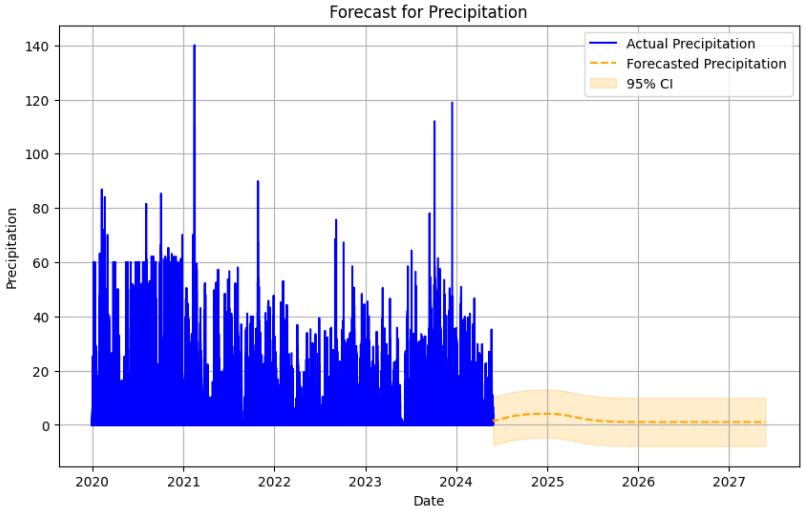
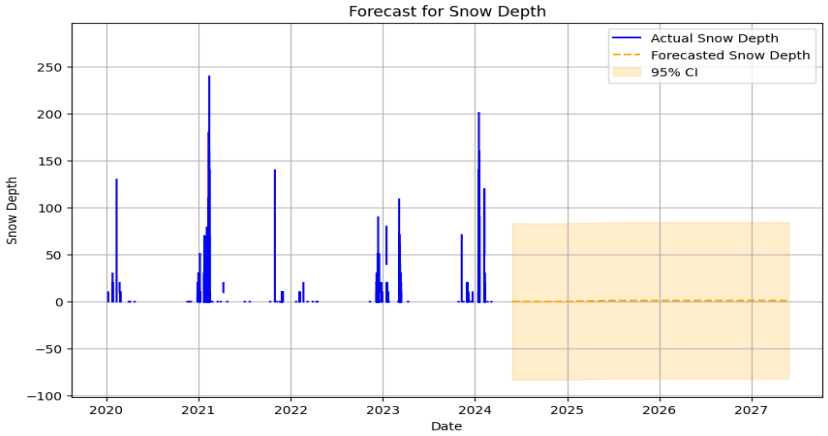


Figure 10:



6.10.4 KEY FORECAST INSIGHTS:

- **Maximum Temperature:** Forecasts indicate a gradual decline in maximum temperature over time, with an increasing degree of uncertainty. The confidence interval widens significantly, reflecting the model's reduced accuracy further into the future.
- **Minimum Temperature:** The model predicts a stable trend in minimum temperatures, with lower uncertainty compared to other parameters. This suggests that the model is more confident in its predictions for minimum temperatures, indicating less variability in this parameter.
- **Precipitation:** The model forecasts a flat trend for precipitation with a relatively large confidence interval. This highlights the challenge in accurately predicting precipitation, given its inherent variability and the complexity of the factors that influence it.
- **Snow Depth:** The forecast for snow depth shows a stable trend, but with a large confidence interval, particularly further into the future. This suggests that while the model can predict trends, the inherent variability in snow events makes precise predictions difficult.

6.10.5 CONCLUSION

The LSTM model, fine-tuned through Bayesian optimization, demonstrated its effectiveness in predicting temperature-related parameters with relatively low error margins. However, it faces challenges in forecasting precipitation and snow depth, reflected in the higher uncertainty for these parameters. The widening confidence intervals in long-term forecasts suggest that while the model captures trends, caution is needed when interpreting these predictions. Future weather-related predictions can be guided by these insights, but the limitations in predicting precipitation and snow depth should be taken into consideration.

ANALYSIS AND DISCUSSION

7.1 OVERVIEW OF MODEL PERFORMANCE AND RESULTS

we take a close look at how well different forecasting models—ARIMA, Prophet, and LSTM—performed in predicting the UK's weather from 2020 to 2024. Each model has its own strengths and weaknesses when it comes to handling time-series data, especially in the complex field of climate forecasting. This analysis will help us understand which models are most effective for predicting specific weather parameters in this context.

7.2 INTERPRETATION OF RESULTS

7.2.1 MODEL COMPARISONS AND EFFECTIVENESS

The LSTM model turned out to be the most accurate for forecasting, showing better performance with lower RMSE and MAE values compared to ARIMA and Prophet. This model's strength lies in its ability to understand the complex, non-linear

relationships within weather data, making it particularly effective for this kind of task. On the other hand, while ARIMA did well with linear trends, it struggled with the more complex, non-linear aspects of the data. Prophet, which works well with strong seasonal patterns, didn't perform as effectively here, likely because of the data's complexity.

The design of the LSTM model, which focuses on handling sequences and long-term dependencies, was crucial in accurately predicting the UK's weather patterns. Its ability to learn from historical trends and predict future outcomes, while considering the data's time-related dependencies, highlights why it's such a robust tool for time-series forecasting.

7.2.2 RELEVANCE TO LITERATURE AND EXISTING MODELS

The performance of the models aligns well with what we've seen in previous studies. LSTM networks have proven to be very effective for forecasting sequential data, especially in areas where patterns change over time, like weather prediction. Our results confirm that LSTM outperforms traditional models such as ARIMA when dealing with non-linear data and long-term trends, which matches findings in earlier research.

On the other hand, ARIMA struggled with the complexity of our data, highlighting the limitations of statistical models when more nuanced understanding of time-based patterns is needed. The Prophet model, while user-friendly and simple, didn't perform as well as expected, likely because our dataset's complexities required a more advanced modelling approach.

7.2.3 IMPLICATIONS FOR CLIMATIC UNDERSTANDING AND PRACTICAL APPLICATIONS

The findings of this study provide valuable insights into climate trends in the UK and their potential impacts. The LSTM model's predictions suggest that temperatures might decline starting in 2024, which could significantly affect agriculture, water resources, and urban planning. The variability and uncertainty in precipitation forecasts highlight the challenges in predicting rainfall patterns, which is crucial for managing water resources and preparing for natural disasters.

The study's outcomes align with the project's goals by offering practical insights into future climate conditions in the UK. Notably, the difference in temperature patterns between urban and rural areas, with higher temperatures in cities, suggests the presence of urban heat island effects—where urbanization leads to higher local temperatures. These insights are important for urban planners and policymakers, emphasizing the need for specific measures to mitigate these effects.

7.2.4 LIMITATIONS OF THE STUDY AND MODEL APPLICATIONS

While the LSTM model demonstrated the highest accuracy, it is important to acknowledge the limitations inherent in all models used. For instance, while the LSTM was successful in predicting temperature trends, its accuracy in forecasting

precipitation was lower, as evidenced by the larger confidence intervals. This limitation suggests that while LSTM is effective for certain types of weather data, other models or hybrid approaches might be necessary to improve precipitation forecasts.

Additionally, the study's reliance on historical data means that any unforeseen climatic shifts or external factors, such as policy changes affecting environmental conditions, might not be fully captured in the models. This limitation should be considered when applying these forecasts to real-world scenarios, particularly in long-term planning and policy development.

7.2.5 PRACTICAL USABILITY OF THE MODELS

In real-world use, the LSTM model's predictions could be incredibly valuable. For example, agricultural planners might use these forecasts to better prepare for future growing conditions, choosing crops that are more likely to thrive. Urban planners could use the temperature predictions to develop cooling strategies that help combat the urban heat island effect. However, because the model is less certain about precipitation forecasts, it would be wise to approach water resource management with caution, acknowledging the potential variability in these predictions.

7.2.6 ADDRESSING THE RESEARCH QUESTIONS

The analysis and model results effectively address the research questions outlined in this report:

- **Variation in Temperature and Precipitation Patterns**: The study identifies significant deviations from historical norms, particularly in urban areas where higher temperatures were consistently observed.
- **Urban vs. Rural Differences:** The clustering and outlier detection analyses highlight the distinct climatic behaviours in urban versus rural areas, with urban regions exhibiting higher temperature patterns.
- **Implications for Agriculture and Water Resources:** The forecasts indicate potential challenges for agricultural productivity due to temperature declines and precipitation variability, emphasizing the need for adaptive strategies in these sectors.

7.3 CONCLUSION

In conclusion, the LSTM model provided the most reliable forecasts among the models tested, particularly for temperature predictions, aligning with the project's objective of identifying and forecasting climatic trends in the UK. While the ARIMA and Prophet models had their strengths, they were less effective in handling the complexities of the dataset. The insights gained from this study are critical for guiding future research and practical applications, especially in the fields of agriculture, water resource management, and urban planning. The limitations

identified suggest areas for further research, such as improving precipitation forecasting and exploring hybrid modelling approaches.

CONCLUSION

This study explored different models—Prophet, ARIMA, and LSTM—to predict weather patterns, focusing on temperature, precipitation, and snow depth. The model stood out, especially for temperature forecasting, thanks to its ability to capture complex, non-linear patterns. However, it struggled more with predicting precipitation and snow depth, reflecting the challenges of these variables.

The LSTM model's strength lies in its application where precise temperature forecasts are crucial, such as in agriculture, urban planning, and energy management. Its ability to process sequential data makes it a valuable tool in these areas. However, improving precipitation forecasting remains a challenge, suggesting the need for more advanced techniques or additional data to enhance accuracy.

These findings have practical implications. For example, accurate temperature forecasts can help farmers optimize planting and irrigation schedules. In cities, they can inform the design of heat-resistant infrastructure and better energy management. Additionally, anticipating temperature trends can improve public health responses to extreme weather events.

Future work could focus on enhancing precipitation forecasts by integrating advanced models or combining LSTM with other machine learning techniques like ensemble learning. Expanding the dataset to cover more diverse climates and testing these models in different regions could also increase their accuracy and reliability. Real-time application of these models in decision-making processes, allowing continuous learning and adaptation, is another promising area for further research.

In summary, while the study has successfully met its goals and provided valuable insights into weather forecasting with time series models, there is still room for improvement, particularly in forecasting precipitation. The findings pave the way for future research and practical applications, with significant potential to impact agriculture, urban planning, and climate change mitigation.

REFERENCES

Hawkins, E. and Sutton, R. (2005). Trend analysis of climate time series: A review omethods.Climatic Change, 70(1-2), pp.21-45. DOI: 10.1007/s10584-005-5936-x.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), pp.1735-1780.

Huang, C. and Petukhina, A. (2022). Modern machine learning methods for time series analysis. In: Applied Time Series Analysis and Forecasting with Python. Statistics and Computing. Cham: Springer. DOI: https://doi.org/10.1007/978-3-031-13584-2_10.

Jones, N. (2017). How machine learning could help to improve climate forecasts. Nature, 548(7668). Available at: https://link.gale.com/apps/doc/A501590693/HRCA?u=anon~6b46ffb&sid=googleScholar&xid=165d230b [Accessed 20 May. 2024].

Khadka, B. (2019). Data analysis theory and practice: Case: Python and Excel tools. Centria University of Applied Sciences. Available at: https://www.theseus.fi/handle/10024/335764 [Accessed 02 May. 2024].

Kolarik, T. and Rudorfer, G. (1994). Time series forecasting using neural networks. In: Proc. Int. Conf. APL (SIGAPL 1994), Antwerp, Belgium, pp.86–94.

National Centers for Environmental Information, (n.d.). Climate monitoring. Available at: https://www.ncei.noaa.gov/monitoring [Accessed 31 May. 2024].

National Oceanic and Atmospheric Administration (NOAA), (n.d.). NOAA Climate Data Online. Available at: https://www.ncdc.noaa.gov/cdo-web/ [Accessed 31 May. 2024].

Pal, A. and Prakash, P.K.S. (2017). Practical Time Series Analysis: Master Time Series Data Processing, Visualization, and Modeling Using Python. Birmingham: Packt Publishing Ltd.

Priyadarshini, I. and Puri, V. (2021). Mars weather data analysis using machine learning techniques. Earth Science Informatics, 14, pp.1885–1898. DOI: https://doi.org/10.1007/s12145-021-00643-0.

Prophet, (2020). Prophet: Forecasting at Scale. Available at: https://facebook.github.io/prophet/ [Accessed 20 Jul. 2024].

Shivanna, K.R. (2022). Climate change and its impact on biodiversity and human welfare. Proceedings of the Indian National Science Academy, 88, pp.160–171. DOI: https://doi.org/10.1007/s43538-022-00073-6.

Shumway, R.H. and Stoffer, D.S. (2017). ARIMA models. In: Time Series Analysis and Its Applications. Springer Texts in Statistics. Cham: Springer. DOI: https://doi.org/10.1007/978-3-319-52452-8_3.

Wu, B. (2021). K-means clustering algorithm and Python implementation. In: Proc. EEE Int. Conf. Comp. Sci., Artif. Intell. & Elec. Eng. (CSAIEE), SC, USA, pp.55-59. DOI: 10.1109/CSAIEE54046.2021.9543260.

Wu, J., Wang, D., Huang, Z., Qi, J. and Wang, R. (2021). Weather temperature prediction based on LSTM-Bayesian optimization. In: X. Sun, X. Zhang, Z. Xia and E.Bertino, eds. Advances in Artificial Intelligence and Security. ICAIS 2021. Communications in Computer and Information Science, vol. 1422. Cham: Springer, pp. 454-466. DOI: https://doi.org/10.1007/978-3-030-78615-1_39.

APPENDICES

GIT URL : https://github.com/vr22abb/MSc-Data-Science-Project

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from scipy.stats import gaussian_kde, kurtosis, skew, norm
from prophet import Prophet
from keras.callbacks import EarlyStopping
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
from kerastuner.tuners import BayesianOptimization
import tensorflow as tf
import warnings
warnings.filterwarnings('ignore')

# Global definitions for column names and units
COLUMNS = {
    'daily': ['TMAX', 'TAVG', 'TMIN', 'PRCP', 'SNWD'],
    'monthly': ['EMXP', 'EMXT', 'DSND', 'PRCP', 'DX90', 'DP10', 'HDSD',
'HTDD', 'DX70', 'DP01', 'CDSD', 'EMNT', 'DT32', 'DT00', 'DX32', 'CLDD',
'TMAX', 'EMSD', 'TAVG', 'TMIN'],
    'yearly': ['EMXP', 'EMXT', 'PRCP', 'DX90', 'DP10', 'DX70', 'EMNT',
'DT32', 'DX32', 'TMAX', 'EMSD', 'TAVG', 'TMIN']
}

COLUMN_NAMES = {
    'daily': ['Maximum Temperature', 'Average Temperature', 'Minimum
Temperature', 'Precipitation', 'Snow Depth'],
    'monthly': ['Extreme Maximum Precipitation', 'Extreme Maximum
Temperature', 'Number of Days with Snow Depth > 1 inch',
```

```
                'Precipitation', 'Number of Days with Maximum
Temperature > 90°F', 'Number of Days with >= 1.0 inch of
Precipitation',
                'Heating Degree Days Season to Date', 'Heating Degree
Days', 'Number of Days with Maximum Temperature > 70°F',
                'Number of Days with >= 0.1 inch of Precipitation',
'Cooling Degree Days Season to Date', 'Extreme Minimum Temperature',
                'Number of Days with Minimum Temperature <= 32°F',
'Number of Days with Minimum Temperature <= 0.0°F',
                'Number of Days with Maximum Temperature < 32°F',
'Cooling Degree Days', 'Maximum Temperature',
                'Extreme Maximum Snow Depth', 'Average Temperature',
'Minimum Temperature'],
    'yearly': ['Extreme Maximum Precipitation', 'Extreme Maximum
Temperature', 'Precipitation', 'Number of Days with Maximum Temperature
> 90°F',
               'Number of Days with >= 1.0 inch of Precipitation',
'Number of Days with Maximum Temperature > 70°F',
               'Extreme Minimum Temperature', 'Number of Days with
Minimum Temperature <= 32°F',
               'Number of Days with Maximum Temperature < 32°F',
'Maximum Temperature', 'Extreme Maximum Snow Depth',
               'Average Temperature', 'Minimum Temperature']
}

UNITS = {
    'daily': ['°C', '°C', '°C', 'mm', 'mm'],
    'monthly': ['mm', '°C', 'days', 'mm', 'days', 'days', 'degree-
days', 'degree-days', 'days', 'days', 'degree-days', '°C', 'days',
'days', 'days', 'degree-days', '°C', 'mm', '°C', '°C'],
    'yearly': ['mm', '°C', 'mm', 'days', 'days', 'days', '°C', 'days',
'days', '°C', 'mm', '°C', '°C']
}

# Global variables to hold datasets
daily_data = None
monthly_data = None
yearly_data = None

def fill_na_with_mean(df):
    """Fills NA values in numeric columns with the column mean."""
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    for col in numeric_cols:
        mean_value = df[col].mean()
        df[col] = df[col].fillna(mean_value)
    return df

def load_dataset(filepath):
    """Loads a dataset from the specified file path and handles
exceptions."""
    try:
        df = pd.read_csv(filepath, parse_dates=['DATE'],
index_col='DATE')
        return df.sort_index()
    except FileNotFoundError as e:
        print(f"Error: {e}")
        return None
    except pd.errors.ParserError as e:
```

```
        print(f"Parsing error: {e}")
        return None
    except Exception as e:
        print(f"Unexpected error: {e}")
        return None

def preprocess_data(df):
    """Preprocesses the data by filling NA values with the mean and
sorting by date."""
    df = fill_na_with_mean(df)
    return df.sort_index()

def load_and_preprocess_data():
    """Loads and preprocesses daily, monthly, and yearly datasets."""
    global daily_data, monthly_data, yearly_data

    daily_data = load_dataset('Daily Summaries - GHCND.csv')
    monthly_data = load_dataset('Global Summary of the Month -
GSOM.csv')
    yearly_data = load_dataset('Global Summary of The Year - GSOY.csv')

    if daily_data is not None:
        daily_data = preprocess_data(daily_data)
    if monthly_data is not None:
        monthly_data = preprocess_data(monthly_data)
    if yearly_data is not None:
        yearly_data = preprocess_data(yearly_data)


def main():
    """Main function to execute the data loading and preprocessing."""
    load_and_preprocess_data()

    if daily_data is None or monthly_data is None or yearly_data is
None:
        print("Data loading failed. Please check the file paths and
formats.")
    else:
        print("Data loaded and preprocessed successfully.")

# Execute the main function
if __name__ == "__main__":
    main()


def aggregate_station_data(df):
    """
    Aggregates data by station, calculating the mean for temperature
parameters (TMAX, TMIN)
    and the sum for precipitation and snow depth (PRCP, SNWD).

    Parameters:
        df (pd.DataFrame): The input DataFrame containing weather data.

    Returns:
        pd.DataFrame: A DataFrame with aggregated data by station.
    """
    return df.groupby(['STATION', 'NAME']).agg({
```

```
        'TMAX': 'mean',
        'TMIN': 'mean',
        'PRCP': 'sum',
        'SNWD': 'sum'
    }).reset_index()

def find_top_stations(df, column, n=5, largest=True):
    """
    Finds the top or lowest n stations based on a specified parameter.

    Parameters:
        df (pd.DataFrame): The input DataFrame with aggregated station
data.
        column (str): The column name on which to base the ranking.
        n (int): The number of stations to return. Default is 5.
        largest (bool): Whether to return the largest values. Default
is True.

    Returns:
        pd.DataFrame: A DataFrame containing the top or lowest n
stations.
    """
    if largest:
        return df.nlargest(n, column)
    else:
        return df.nsmallest(n, column)

def plot_top_stations(df, column, title, xlabel, ylabel):
    """
    Plots the top or lowest n stations based on a specified parameter.

    Parameters:
        df (pd.DataFrame): The input DataFrame with top or lowest
stations data.
        column (str): The column name to plot.
        title (str): The title of the plot.
        xlabel (str): The x-axis label.
        ylabel (str): The y-axis label.
    """
    plt.figure(figsize=(10, 6))
    plt.bar(df['NAME'], df[column], color=plt.cm.tab20.colors)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.title(title)
    plt.xticks(rotation=45, ha='right')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()

def display_and_plot_top_stations(df):
    """
    Displays and plots the top and lowest 5 stations based on different
weather parameters.

    Parameters:
        df (pd.DataFrame): The input DataFrame with aggregated station
data.
    """
    top5_tmax = find_top_stations(df, 'TMAX', 5, largest=True)
```

```
    top5_tmin = find_top_stations(df, 'TMIN', 5, largest=True)
    top5_prcp = find_top_stations(df, 'PRCP', 5, largest=True)
    top5_snwd = find_top_stations(df, 'SNWD', 5, largest=True)
    lowest5_prcp = find_top_stations(df, 'PRCP', 5, largest=False)
    lowest5_snwd = find_top_stations(df, 'SNWD', 5, largest=False)

    print("Top 5 Stations based on the highest average maximum
temperature (TMAX):")
    print(top5_tmax.to_string(index=False))
    plot_top_stations(top5_tmax, 'TMAX', 'Top 5 Stations with Highest
TMAX', 'Station', 'TMAX (°C)')

    print("\nTop 5 Stations based on the highest average minimum
temperature (TMIN):")
    print(top5_tmin.to_string(index=False))
    plot_top_stations(top5_tmin, 'TMIN', 'Top 5 Stations with Highest
TMIN', 'Station', 'TMIN (°C)')

    print("\nTop 5 Stations based on the highest total precipitation
(PRCP):")
    print(top5_prcp.to_string(index=False))
    plot_top_stations(top5_prcp, 'PRCP', 'Top 5 Stations with Highest
PRCP', 'Station', 'PRCP (mm)')

    print("\nTop 5 Stations based on the highest total snow depth
(SNWD):")
    print(top5_snwd.to_string(index=False))
    plot_top_stations(top5_snwd, 'SNWD', 'Top 5 Stations with Highest
SNWD', 'Station', 'SNWD (mm)')

    print("\nTop 5 Stations based on the lowest total precipitation
(PRCP):")
    print(lowest5_prcp.to_string(index=False))
    plot_top_stations(lowest5_prcp, 'PRCP', 'Top 5 Stations with Lowest
PRCP', 'Station', 'PRCP (mm)')

    print("\nTop 5 Stations based on the lowest total snow depth
(SNWD):")
    print(lowest5_snwd.to_string(index=False))
    plot_top_stations(lowest5_snwd, 'SNWD', 'Top 5 Stations with Lowest
SNWD', 'Station', 'SNWD (mm)')

def main():
    """
    Main function to execute the station data aggregation and display
the top and lowest stations.
    """
    station_aggregated_data = aggregate_station_data(daily_data)
    display_and_plot_top_stations(station_aggregated_data)

if __name__ == "__main__":
    main()


def display_summary_statistics(df, name, columns):
    """
    Displays summary statistics for the specified columns of a
DataFrame.
```

```
    Parameters:
        df (pd.DataFrame): The DataFrame containing the data.
        name (str): A name for the data to be displayed.
        columns (list): A list of columns for which to display summary
statistics.
    """
    if df.empty:
        print(f"No data available for {name}.")
    else:
        print(f"\nSummary Statistics for {name}:\n")
        print(df[columns].describe())

def display_statistics_for_all_datasets(daily_data, monthly_data,
yearly_data, columns):
    """
    Displays summary statistics for daily, monthly, and yearly datasets
based on global definitions.

    Parameters:
        daily_data (pd.DataFrame): The daily dataset.
        monthly_data (pd.DataFrame): The monthly dataset.
        yearly_data (pd.DataFrame): The yearly dataset.
        columns (dict): A dictionary containing column definitions for
each dataset.
    """
    display_summary_statistics(daily_data, 'Daily Data',
columns['daily'])
    display_summary_statistics(monthly_data, 'Monthly Data',
columns['monthly'])
    display_summary_statistics(yearly_data, 'Yearly Data',
columns['yearly'])

def main():
    """
    Main function to execute the summary statistics display for all
datasets.
    """
    display_statistics_for_all_datasets(daily_data, monthly_data,
yearly_data, COLUMNS)

if __name__ == "__main__":
    main()


def plot_histogram_and_kde(ax, data, col_name, bin_width):
    """
    Plots the histogram and KDE for a specific column of data.

    Parameters:
        ax (matplotlib.axes.Axes): The axes object to plot on.
        data (pd.Series): The data to plot.
        col_name (str): The name of the column being plotted.
        bin_width (float): The width of the bins for the histogram.
    """
    bins = np.arange(data.min(), data.max() + bin_width, bin_width)
    ax.hist(data, bins=bins, density=True, alpha=0.7, color='skyblue',
edgecolor='black')
```

```
    if len(data.unique()) > 1:
        kde = gaussian_kde(data)
        x = np.linspace(data.min(), data.max(), 1000)
        ax.plot(x, kde(x), color='blue')

    mean = data.mean()
    std_dev = data.std()
    xmin, xmax = ax.get_xlim()
    x = np.linspace(xmin, xmax, 1000)
    p = norm.pdf(x, mean, std_dev)
    ax.plot(x, p, color='red', linestyle='dashed')

def add_statistics_text(ax, data, col_name):
    """
    Adds statistical annotations to the plot.

    Parameters:
        ax (matplotlib.axes.Axes): The axes object to add text to.
        data (pd.Series): The data to calculate statistics for.
        col_name (str): The name of the column being analyzed.
    """
    mean = data.mean()
    std_dev = data.std()
    skewness = skew(data)
    kurt = kurtosis(data)
    sample_size = len(data)

    stats_text = (f'Mean: {mean:.2f}\n'
                  f'STD: {std_dev:.2f}\n'
                  f'Skew: {skewness:.2f}\n'
                  f'Kurtosis: {kurt:.2f}\n'
                  f'Sample size: {sample_size}')
    ax.text(0.95, 0.95, stats_text, transform=ax.transAxes,
            verticalalignment='top', horizontalalignment='right',
            fontsize=10, bbox=dict(boxstyle='round,pad=0.5',
edgecolor='gray', facecolor='white'))

def plot_histograms_with_stats(df, name, cols, col_names, units,
figsize, suptitle_y):
    """
    Plots histograms with statistics for a given dataset.

    Parameters:
        df (pd.DataFrame): The DataFrame containing the data.
        name (str): The name of the dataset (e.g., 'Daily', 'Monthly',
'Yearly').
        cols (list): List of column names to plot.
        col_names (list): List of column display names for the plots.
        units (list): List of units for each column.
        figsize (tuple): Figure size for the plot.
        suptitle_y (float): Y position for the subplot title.
    """
    num_rows = len(cols)
    fig, axes = plt.subplots(num_rows, 1, figsize=figsize,
sharex=False)
    fig.suptitle(f'Histograms of {name} Data with Statistics',
fontsize=20, y=suptitle_y)
```

```
    for i, (col, col_name, unit) in enumerate(zip(cols, col_names,
units)):
        data = pd.to_numeric(df[col], errors='coerce').dropna()
        ax = axes[i] if num_rows > 1 else axes

        if not data.empty:
            bin_width = (data.max() - data.min()) / 30
            plot_histogram_and_kde(ax, data, col_name, bin_width)
            add_statistics_text(ax, data, col_name)

            ax.set_title(col_name, fontsize=14)
            ax.set_ylabel('Density', fontsize=12)
            ax.set_xlabel(f'{col_name} ({unit})', fontsize=12)
            ax.grid(True, linestyle='--', alpha=0.7)
        else:
            ax.set_title(f"{col_name} - No Data Available")
            ax.axis('off')

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.subplots_adjust(top=0.93, hspace=0.5)
    plt.show()

def plot_histograms_for_dataset(daily_data, monthly_data, yearly_data,
columns, column_names, units):
    """
    Plots histograms with statistics for daily, monthly, and yearly
datasets.

    Parameters:
        daily_data (pd.DataFrame): The daily dataset.
        monthly_data (pd.DataFrame): The monthly dataset.
        yearly_data (pd.DataFrame): The yearly dataset.
        columns (dict): Dictionary of columns for each dataset.
        column_names (dict): Dictionary of column display names for
each dataset.
        units (dict): Dictionary of units for each dataset.
    """
    daily_figsize = (10, len(columns['daily']) * 3)
    plot_histograms_with_stats(daily_data, 'Daily', columns['daily'],
column_names['daily'], units['daily'], daily_figsize, suptitle_y=1)

    monthly_figsize = (10, len(columns['monthly']) * 3)
    plot_histograms_with_stats(monthly_data, 'Monthly',
columns['monthly'], column_names['monthly'], units['monthly'],
monthly_figsize, suptitle_y=0.95)

    yearly_figsize = (10, len(columns['yearly']) * 3)
    plot_histograms_with_stats(yearly_data, 'Yearly',
columns['yearly'], column_names['yearly'], units['yearly'],
yearly_figsize, suptitle_y=0.95)

def main():
    """
    Main function to execute histogram plotting for all datasets.
    """
    plot_histograms_for_dataset(daily_data, monthly_data, yearly_data,
COLUMNS, COLUMN_NAMES, UNITS)
```

```
if __name__ == "__main__":
    main()


def categorize_season(date):
    """
    Categorizes a given date into a season (Winter, Spring, Summer,
Fall).
    """
    month = date.month
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    else:
        return 'Fall'

def plot_seasonal_trends(df, title):
    """
    Plots seasonal trends for temperature, precipitation, and snow
depth.
    """
    fig, ax1 = plt.subplots(figsize=(14, 7))
    color1, color2, color3, color4 = 'tab:blue', 'tab:green',
'tab:red', 'tab:purple'

    ax1.set_xlabel('Season')
    ax1.set_ylabel('Temperature (°C)', color=color1)
    ax1.plot(df.index, df['TMAX'], color=color1, label='TMAX',
marker='o')
    ax1.plot(df.index, df['TMIN'], color=color3, label='TMIN',
marker='s')
    ax1.tick_params(axis='y', labelcolor=color1)

    ax2 = ax1.twinx()
    ax2.set_ylabel('Precipitation (mm)', color=color2)
    ax2.plot(df.index, df['PRCP'], color=color2, label='PRCP',
marker='^')
    ax2.tick_params(axis='y', labelcolor=color2)

    if 'SNWD' in df.columns:
        ax3 = ax1.twinx()
        ax3.spines['right'].set_position(('outward', 60))
        ax3.set_ylabel('Snow Depth (mm)', color=color4)
        ax3.plot(df.index, df['SNWD'], color=color4, label='SNWD',
marker='x')
        ax3.tick_params(axis='y', labelcolor=color4)
        ax3.legend(loc='upper center')

    fig.suptitle(title)
    fig.tight_layout()
    ax1.legend(loc='upper left')
    ax2.legend(loc='upper right')
    plt.grid(True)
    plt.show()
```

```
def calculate_monthly_stats(df, columns):
    """
    Resamples data to monthly frequency and calculates mean and
standard deviation for each parameter.
    """
    monthly_data = df[columns].resample('M').mean()
    monthly_data['Month'] = monthly_data.index.month
    monthly_stats = monthly_data.groupby('Month').agg({col: ['mean',
'std'] for col in columns})
    monthly_stats.columns = [f'{col}_{stat}' for col, stat in
monthly_stats.columns]
    return monthly_stats

def plot_combined_monthly_variability(stats, title):
    """
    Plots monthly variability with error bars for temperature,
precipitation, and snow depth.
    """
    fig, ax1 = plt.subplots(figsize=(14, 7))
    months = range(1, 13)

    ax1.errorbar(months, stats['TMAX_mean'], yerr=stats['TMAX_std'],
fmt='-o', color='tab:blue', label='TMAX')
    ax1.errorbar(months, stats['TMIN_mean'], yerr=stats['TMIN_std'],
fmt='-s', color='tab:red', label='TMIN')
    ax1.set_xlabel('Month')
    ax1.set_ylabel('Temperature (°C)')
    ax1.legend(loc='upper left')
    ax1.grid(True)

    ax2 = ax1.twinx()
    ax2.errorbar(months, stats['PRCP_mean'], yerr=stats['PRCP_std'],
fmt='-^', color='tab:green', label='Precipitation')
    ax2.errorbar(months, stats['SNWD_mean'], yerr=stats['SNWD_std'],
fmt='-x', color='tab:purple', label='Snow Depth')
    ax2.set_ylabel('Values (mm)')
    ax2.legend(loc='upper right')

    plt.xticks(months, ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
    plt.title(title)
    plt.show()

def calculate_yearly_stats(df, columns):
    """
    Resamples data to yearly frequency and calculates mean and standard
deviation for each parameter.
    """
    yearly_data = df[columns].resample('Y').mean()
    yearly_data['Year'] = yearly_data.index.year
    yearly_stats = yearly_data.groupby('Year').agg(['mean', 'std'])
    yearly_stats.columns = ['_'.join(col) for col in
yearly_stats.columns]
    return yearly_stats

def plot_yearly_variability(stats, title):
    """
```

```
    Plots yearly variability with error bars for temperature,
precipitation, and snow depth.
    """
    fig, ax1 = plt.subplots(figsize=(14, 7))
    years = stats.index
    custom_ticks = [f"{year}.0" for year in years]

    ax1.errorbar(custom_ticks, stats['TMAX_mean'],
yerr=stats['TMAX_std'], fmt='-o', color='tab:blue', label='TMAX')
    ax1.errorbar(custom_ticks, stats['TMIN_mean'],
yerr=stats['TMIN_std'], fmt='-s', color='tab:red', label='TMIN')
    ax1.set_xlabel('Year')
    ax1.set_ylabel('Temperature (°C)')
    ax1.legend(loc='upper left')
    ax1.grid(True)

    ax2 = ax1.twinx()
    ax2.errorbar(custom_ticks, stats['PRCP_mean'],
yerr=stats['PRCP_std'], fmt='-^', color='tab:green',
label='Precipitation')
    ax2.errorbar(custom_ticks, stats['SNWD_mean'],
yerr=stats['SNWD_std'], fmt='-x', color='tab:purple', label='Snow
Depth')
    ax2.set_ylabel('Values (mm)')
    ax2.legend(loc='upper right')

    plt.title(title)
    plt.xticks(range(0, len(custom_ticks)), custom_ticks, rotation=45)
    plt.show()

def load_and_prepare_data(filepath):
    """
    Loads weather data from a CSV file and categorizes the season for
each date.
    """
    df = pd.read_csv(filepath, parse_dates=['DATE'], index_col='DATE')
    df = df.sort_index()
    df['Season'] = df.index.map(categorize_season)
    return df

def main():
    """
    Main function to load data, calculate statistics, and plot trends
and variability.
    """
    # Load the data
    daily_data = load_and_prepare_data('Daily Summaries - GHCND.csv')

    # Aggregate data by season
    daily_data_numeric = daily_data.select_dtypes(include=[np.number])
    daily_data_numeric['Season'] = daily_data['Season']
    seasonal_data = daily_data_numeric.groupby('Season').mean()

    # Plot seasonal trends
    plot_seasonal_trends(seasonal_data, 'Seasonal Trends of Maximum
Temperature, Minimum Temperature, Precipitation, and Snow Depth')

    # Columns to analyze
```

```
    columns = ['TMAX', 'TMIN', 'PRCP', 'SNWD']

    # Calculate and plot monthly statistics
    monthly_stats = calculate_monthly_stats(daily_data, columns)
    plot_combined_monthly_variability(monthly_stats, 'Monthly
Temperature, Precipitation, and Snow Depth Variability')

    # Calculate and plot yearly statistics
    yearly_stats = calculate_yearly_stats(daily_data, columns)
    plot_yearly_variability(yearly_stats, 'Yearly Temperature,
Precipitation, and Snow Depth Variability')

if __name__ == "__main__":
    main()


def select_and_dropna(df, columns):
    """
    Selects specific columns from a DataFrame and drops rows with NaN
values.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        columns (list): The list of columns to select.

    Returns:
        pd.DataFrame: A DataFrame with the selected columns and NaN
values dropped.
    """
    return df[columns].dropna()

def aggregate_data(df, frequency):
    """
    Aggregates data by the specified frequency (e.g., monthly or
yearly).

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        frequency (str): The frequency for resampling (e.g., 'M' for
monthly, 'Y' for yearly).

    Returns:
        pd.DataFrame: A DataFrame with aggregated data by the specified
frequency.
    """
    return df.resample(frequency).mean()

def plot_correlation_matrix(data, title, column_names):
    """
    Plots the correlation matrix with correlation values displayed.

    Parameters:
        data (pd.DataFrame): The input DataFrame.
        title (str): The title for the plot.
        column_names (list): The list of column names to display in the
plot.
    """
    correlation_matrix = data.corr()
```

```
    plt.figure(figsize=(10, 8))
    plt.matshow(correlation_matrix, cmap='coolwarm', fignum=1)
    plt.colorbar()

    # Display correlation values in the matrix
    for (i, j), val in np.ndenumerate(correlation_matrix.values):
        plt.text(j, i, f'{val:.2f}', ha='center', va='center',
color='black')

    # Add labels with column names
    plt.xticks(range(len(column_names)), column_names, rotation=45)
    plt.yticks(range(len(column_names)), column_names)
    plt.title(title, pad=20)
    plt.show()

def main():
    """
    Main function to perform data selection, aggregation, and
correlation matrix plotting.
    """
    # Assuming daily_data is already loaded and preprocessed
    selected_columns = ['TMIN', 'TMAX', 'PRCP', 'SNWD']
    column_names = ['Minimum Temperature', 'Maximum Temperature',
'Precipitation', 'Snow Depth']

    # Select specific columns and drop NaN values
    filtered_data = select_and_dropna(daily_data, selected_columns)

    # Aggregate data by month and year
    monthly_aggregated_data = aggregate_data(filtered_data, 'M')
    yearly_aggregated_data = aggregate_data(filtered_data, 'Y')

    # Plot correlation matrix for daily, monthly, and yearly data
    plot_correlation_matrix(filtered_data, 'Correlation Matrix for
Daily Weather Data', column_names)
    plot_correlation_matrix(monthly_aggregated_data, 'Correlation
Matrix for Monthly Aggregated Weather Data', column_names)
    plot_correlation_matrix(yearly_aggregated_data, 'Correlation Matrix
for Yearly Aggregated Weather Data', column_names)

if __name__ == "__main__":
    main()


def prepare_data(df, columns):
    """
    Prepares the data by selecting specific columns and scaling the
values.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        columns (list): The list of columns to select.

    Returns:
        np.ndarray: Scaled data array.
    """
    X = df[columns].dropna()
```

```
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled

def apply_pca(data, n_components=2):
    """
    Applies PCA transformation to reduce the dimensionality of the
data.

    Parameters:
        data (np.ndarray): The input scaled data.
        n_components (int): The number of principal components to keep.

    Returns:
        np.ndarray: Transformed data with reduced dimensions.
    """
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(data)
    return X_pca

def apply_kmeans(data, n_clusters=3, random_state=42):
    """
    Applies K-means clustering to the data.

    Parameters:
        data (np.ndarray): The input scaled data.
        n_clusters (int): The number of clusters for K-means.
        random_state (int): The random state for reproducibility.

    Returns:
        np.ndarray: Cluster labels for each data point.
    """
    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
    clusters = kmeans.fit_predict(data)
    return clusters

def plot_clusters(X_pca, clusters, title):
    """
    Plots the clusters on a 2D PCA-transformed space.

    Parameters:
        X_pca (np.ndarray): PCA-transformed data.
        clusters (np.ndarray): Cluster labels for each data point.
        title (str): The title of the plot.
    """
    plt.figure(figsize=(10, 8))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis',
marker='o', edgecolor='k')
    plt.title(title)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.colorbar(label='Cluster')
    plt.show()

def main():
    """
    Main function to execute the data preparation, PCA, K-means
clustering, and plotting.
```

```
    """
    columns = ['TMAX', 'TMIN', 'PRCP', 'SNWD']

    # Prepare the data
    X_scaled = prepare_data(daily_data, columns)

    # Apply PCA
    X_pca = apply_pca(X_scaled)

    # Apply K-means clustering
    clusters = apply_kmeans(X_scaled)

    # Plot the clusters
    plot_clusters(X_pca, clusters, 'K-means Clustering of Daily Weather
Data')

if __name__ == "__main__":
    main()


def detect_outliers_iqr(df, column):
    """
    Detects outliers in a DataFrame column using the Interquartile
Range (IQR) method.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        column (str): The column name to detect outliers in.

    Returns:
        pd.DataFrame: A DataFrame containing only the outliers.
    """
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df[column] < lower_bound) | (df[column] >
upper_bound)]
    return outliers

def plot_outliers_iqr(df, outliers, column, title, y_label):
    """
    Plots the original data and highlights the outliers.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        outliers (pd.DataFrame): The DataFrame containing outliers.
        column (str): The column to plot.
        title (str): The title of the plot.
        y_label (str): The label for the y-axis.
    """
    fig, ax = plt.subplots(figsize=(14, 7))
    ax.plot(df.index, df[column], label=column, color='blue',
linestyle='--', linewidth=0.75, alpha=0.6)
    ax.scatter(outliers.index, outliers[column], color='red',
label='Outlier', edgecolor='k', zorder=5, alpha=0.8)
    ax.set_title(title)
```

```
    ax.set_xlabel('Date')
    ax.set_ylabel(y_label)
    ax.legend()
    ax.grid(True)
    plt.show()

def summarize_outliers(df, outliers, column):
    """
    Summarizes the statistics of detected outliers in a DataFrame
column.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        outliers (pd.DataFrame): The DataFrame containing outliers.
        column (str): The column to summarize.

    Returns:
        dict: A dictionary containing the summary statistics of the
outliers.
    """
    summary = {
        'column': column,
        'num_outliers': len(outliers),
        'mean_outliers': outliers[column].mean(),
        'std_outliers': outliers[column].std(),
        'min_outlier': outliers[column].min(),
        'max_outlier': outliers[column].max(),
    }
    return summary

def print_outlier_summary(summary):
    """
    Prints the summary of outliers in a readable format.

    Parameters:
        summary (dict): The summary statistics of the outliers.
    """
    print(f"Column: {summary['column']}")
    print(f"Number of Outliers: {summary['num_outliers']}")
    print(f"Mean of Outliers: {summary['mean_outliers']:.2f}")
    print(f"Standard Deviation of Outliers:
{summary['std_outliers']:.2f}")
    print(f"Minimum Outlier Value: {summary['min_outlier']:.2f}")
    print(f"Maximum Outlier Value: {summary['max_outlier']:.2f}")
    print("\n")

def visualize_outlier_summary(summaries):
    """
    Visualizes the summary statistics of outliers using bar charts.

    Parameters:
        summaries (list): A list of dictionaries containing summary
statistics of outliers.
    """
    columns = [summary['column'] for summary in summaries]
    num_outliers = [summary['num_outliers'] for summary in summaries]
    mean_outliers = [summary['mean_outliers'] for summary in summaries]
    std_outliers = [summary['std_outliers'] for summary in summaries]
```

```
    fig, axs = plt.subplots(3, 1, figsize=(12, 18))

    # Number of Outliers
    axs[0].bar(columns, num_outliers, color='tab:blue')
    axs[0].set_title('Number of Outliers')
    axs[0].set_xlabel('Parameters')
    axs[0].set_ylabel('Count')

    # Mean of Outliers
    axs[1].bar(columns, mean_outliers, color='tab:green')
    axs[1].set_title('Mean of Outliers')
    axs[1].set_xlabel('Parameters')
    axs[1].set_ylabel('Mean Value')

    # Standard Deviation of Outliers
    axs[2].bar(columns, std_outliers, color='tab:red')
    axs[2].set_title('Standard Deviation of Outliers')
    axs[2].set_xlabel('Parameters')
    axs[2].set_ylabel('Standard Deviation')

    plt.tight_layout()
    plt.show()

def process_and_visualize_outliers(df, columns, column_names):
    """
    Processes and visualizes outliers for a list of columns in a
DataFrame.

    Parameters:
        df (pd.DataFrame): The input DataFrame.
        columns (list): The list of column names to process.
        column_names (list): The list of column names for display in
plots.
    """
    outlier_summaries = []

    for column, column_name in zip(columns, column_names):
        outliers = detect_outliers_iqr(df, column)
        if not outliers.empty:
            plot_outliers_iqr(df, outliers, column, f'Outliers in
{column_name} using IQR', column_name)
            summary = summarize_outliers(df, outliers, column)
            outlier_summaries.append(summary)
            print_outlier_summary(summary)

    # Visualize the outlier summary if there are any summaries
    if outlier_summaries:
        visualize_outlier_summary(outlier_summaries)
    else:
        print("No outliers detected in the specified columns using
IQR.")

def main():
    """
    Main function to process and visualize outliers for the daily
weather data.
    """
```

```
    columns = ['TMAX', 'TMIN', 'PRCP', 'SNWD']
    column_names = ['Maximum Temperature', 'Minimum Temperature',
'Precipitation', 'Snow Depth']
    process_and_visualize_outliers(daily_data, columns, column_names)

if __name__ == "__main__":
    main()


def load_and_preprocess_data(filepath):
    """
    Loads and preprocesses weather data from a CSV file.

    Parameters:
    filepath (str): The path to the CSV file.

    Returns:
    pd.DataFrame: Preprocessed DataFrame with missing numeric values
filled.
    """
    df = pd.read_csv(filepath, parse_dates=['DATE'], index_col='DATE')
    df = df.sort_index()
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
    return df

def prepare_data_for_prophet(df, column):
    """
    Prepares data for Prophet model training.

    Parameters:
    df (pd.DataFrame): The input DataFrame.
    column (str): The column to be forecasted.

    Returns:
    pd.DataFrame: DataFrame formatted for Prophet with columns 'ds'
(dates) and 'y' (values).
    """
    df_prophet = df[[column]].reset_index()
    df_prophet.columns = ['ds', 'y']
    return df_prophet

def train_test_split_prophet(df, test_size=0.2):
    """
    Splits the data into training and test sets for Prophet.

    Parameters:
    df (pd.DataFrame): The DataFrame formatted for Prophet.
    test_size (float): The proportion of data to include in the test
set.

    Returns:
    tuple: Training and test datasets.
    """
    train_size = int(len(df) * (1 - test_size))
    train_data = df.iloc[:train_size]
    test_data = df.iloc[train_size:]
    return train_data, test_data
```

```
def train_prophet_model(train_data):
    """
    Trains a Prophet model on the provided training data.

    Parameters:
    train_data (pd.DataFrame): The training data formatted for Prophet.

    Returns:
    Prophet: A trained Prophet model.
    """
    model = Prophet(daily_seasonality=True, yearly_seasonality=True)
    model.fit(train_data)
    return model

def forecast_with_prophet(model, test_data):
    """
    Uses a trained Prophet model to forecast future values.

    Parameters:
    model (Prophet): A trained Prophet model.
    test_data (pd.DataFrame): The test data to forecast on.

    Returns:
    np.array: The predicted values.
    """
    future = model.make_future_dataframe(periods=len(test_data),
freq='D')
    forecast = model.predict(future)
    y_pred = forecast['yhat'].iloc[-len(test_data):].values
    return y_pred

def evaluate_prophet_model(y_true, y_pred):
    """
    Evaluates the Prophet model using RMSE and MAE.

    Parameters:
    y_true (np.array): The true values.
    y_pred (np.array): The predicted values.

    Returns:
    tuple: RMSE and MAE values.
    """
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    mae = mean_absolute_error(y_true, y_pred)
    return rmse, mae

def run_prophet_pipeline(filepath, column, test_size=0.2):
    """
    Runs the entire Prophet modeling pipeline from data loading to
evaluation.

    Parameters:
    filepath (str): The path to the CSV file.
    column (str): The column to be forecasted.
    test_size (float): The proportion of data to include in the test
set.
```

```
    Returns:
    tuple: RMSE and MAE values for the Prophet model.
    """
    # Load and preprocess data
    df = load_and_preprocess_data(filepath)

    # Prepare data for Prophet
    df_prophet = prepare_data_for_prophet(df, column)

    # Split data into training and test sets
    train_data, test_data = train_test_split_prophet(df_prophet,
test_size=test_size)

    # Train Prophet model
    model = train_prophet_model(train_data)

    # Forecast using the Prophet model
    y_pred = forecast_with_prophet(model, test_data)

    # Evaluate the model
    prophet_rmse, prophet_mae =
evaluate_prophet_model(test_data['y'].values, y_pred)

    # Return evaluation results
    return prophet_rmse, prophet_mae

# Run the Prophet pipeline and store the results
prophet_rmse, prophet_mae = run_prophet_pipeline('Daily Summaries -
GHCND.csv', 'TMAX')

# Print evaluation results
print(f'Prophet RMSE: {prophet_rmse:.2f}')
print(f'Prophet MAE: {prophet_mae:.2f}')


def load_and_preprocess_data(filepath):
    """
    Loads and preprocesses weather data from a CSV file.

    Parameters:
    filepath (str): The path to the CSV file.

    Returns:
    pd.DataFrame: Preprocessed DataFrame with missing numeric values
filled.
    """
    df = pd.read_csv(filepath, parse_dates=['DATE'], index_col='DATE')
    df = df.sort_index()
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
    return df

def evaluate_arima_model(train_data, test_data, order=(5, 1, 0)):
    """
    Evaluates an ARIMA model on the given train and test data.

    Parameters:
    train_data (pd.Series): The training data.
```

```
    test_data (pd.Series): The test data.
    order (tuple): The (p, d, q) order of the ARIMA model.

    Returns:
    tuple: RMSE, MAE, predictions, and true test values.
    """
    # Fit ARIMA model
    model = ARIMA(train_data, order=order)
    model_fit = model.fit()

    # Forecast
    y_pred = model_fit.forecast(steps=len(test_data))

    # Evaluate the predictions
    rmse = np.sqrt(mean_squared_error(test_data, y_pred))
    mae = mean_absolute_error(test_data, y_pred)

    return rmse, mae, y_pred, test_data

def run_arima_pipeline(filepath, column, order=(5, 1, 0),
test_size=0.2):
    """
    Runs the ARIMA model evaluation pipeline from data loading to
evaluation.

    Parameters:
    filepath (str): The path to the CSV file.
    column (str): The column to be forecasted.
    order (tuple): The (p, d, q) order of the ARIMA model.
    test_size (float): The proportion of data to include in the test
set.

    Returns:
    tuple: RMSE and MAE values for the ARIMA model.
    """
    # Load and preprocess data
    df = load_and_preprocess_data(filepath)

    # Split the data into training and test sets
    data = df[column]
    train_size = int(len(data) * (1 - test_size))
    train_data, test_data = data[:train_size], data[train_size:]

    # Evaluate ARIMA model
    arima_rmse, arima_mae, y_pred, y_true =
evaluate_arima_model(train_data, test_data, order=order)

    # Return evaluation results
    return arima_rmse, arima_mae

# Run the ARIMA pipeline and store the results
arima_rmse, arima_mae = run_arima_pipeline('Daily Summaries -
GHCND.csv', 'TMAX', order=(5, 1, 0))

print(f'ARIMA RMSE: {arima_rmse}')
print(f'ARIMA MAE: {arima_mae}')

def load_and_preprocess_data(filepath):
```

```
    """
    Loads and preprocesses weather data from a CSV file.

    Parameters:
    filepath (str): The path to the CSV file.

    Returns:
    pd.DataFrame: Preprocessed DataFrame with normalized values and
added time-related features.
    """
    daily_data = pd.read_csv(filepath, parse_dates=['DATE'],
index_col='DATE')
    daily_data = daily_data.sort_index()

    # Select specific columns and handle missing values by filling with
median
    columns = ['TMAX', 'TMIN', 'PRCP', 'SNWD']
    data = daily_data[columns].apply(pd.to_numeric, errors='coerce')
    data = data.fillna(data.median())

    # Add time-related features
    data['Month'] = data.index.month
    data['Day'] = data.index.day
    data['Year'] = data.index.year
    data['Season'] = (data.index.month % 12 // 3 + 1).astype(str) +
data.index.year.astype(str)

    # Normalize the data
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(data)

    return scaled_data, scaler

def train_test_split(data, test_size=0.2):
    """
    Splits the data into training and testing sets.

    Parameters:
    data (np.array): The preprocessed and normalized data.
    test_size (float): The proportion of the data to include in the
test set.

    Returns:
    tuple: Training and testing sets.
    """
    train_size = int(len(data) * (1 - test_size))
    train, test = data[:train_size], data[train_size:]
    return train, test

def create_sequences(data, seq_length=30):
    """
    Creates sequences of data for LSTM input.

    Parameters:
    data (np.array): The data to create sequences from.
    seq_length (int): The length of each sequence.

    Returns:
```

```
    tuple: Arrays of input sequences (X) and corresponding targets (y).
    """
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length, 0])  # Predicting TMAX as an
example
    return np.array(X), np.array(y)

def build_and_train_lstm_model(X_train, y_train, input_shape,
epochs=20, batch_size=32):
    """
    Builds and trains an LSTM model.

    Parameters:
    X_train (np.array): The training input data.
    y_train (np.array): The training target data.
    input_shape (tuple): The shape of the input data.
    epochs (int): Number of epochs for training.
    batch_size (int): Batch size for training.

    Returns:
    Sequential: A trained LSTM model.
    """
    model = Sequential()
    model.add(LSTM(50, return_sequences=True, input_shape=input_shape))
    model.add(Dropout(0.2))
    model.add(LSTM(50))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')

    model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
validation_split=0.1, verbose=1)
    return model

def evaluate_lstm_model(model, X_test, y_test, scaler, num_features):
    """
    Evaluates the LSTM model and returns RMSE and MAE.

    Parameters:
    model (Sequential): The trained LSTM model.
    X_test (np.array): The test input data.
    y_test (np.array): The actual test target data.
    scaler (MinMaxScaler): The scaler used for normalizing the data.
    num_features (int): Number of features in the dataset.

    Returns:
    tuple: RMSE and MAE values, predicted values, and actual values.
    """
    y_pred = model.predict(X_test)
    y_pred = scaler.inverse_transform(np.concatenate((y_pred,
np.zeros((y_pred.shape[0], num_features - 1))), axis=1))[:, 0]
    y_test_actual =
scaler.inverse_transform(np.concatenate((y_test.reshape(-1, 1),
np.zeros((y_test.shape[0], num_features - 1))), axis=1))[:, 0]

    lstm_rmse = np.sqrt(mean_squared_error(y_test_actual, y_pred))
    lstm_mae = mean_absolute_error(y_test_actual, y_pred)
```

```
    return lstm_rmse, lstm_mae, y_pred, y_test_actual

def run_lstm_pipeline(filepath, seq_length=30, test_size=0.2,
epochs=20, batch_size=32):
    """
    Runs the entire LSTM forecasting pipeline.

    Parameters:
    filepath (str): The path to the CSV file.
    seq_length (int): The length of sequences for LSTM input.
    test_size (float): The proportion of the data to include in the
test set.
    epochs (int): Number of epochs for training.
    batch_size (int): Batch size for training.

    Returns:
    tuple: RMSE and MAE values for the LSTM model.
    """
    # Load and preprocess data
    scaled_data, scaler = load_and_preprocess_data(filepath)

    # Split the data into train and test sets
    train, test = train_test_split(scaled_data, test_size)

    # Create sequences for LSTM
    X_train, y_train = create_sequences(train, seq_length)
    X_test, y_test = create_sequences(test, seq_length)

    # Build and train LSTM model
    model = build_and_train_lstm_model(X_train, y_train,
input_shape=(X_train.shape[1], X_train.shape[2]), epochs=epochs,
batch_size=batch_size)

    # Evaluate the model
    lstm_rmse, lstm_mae, y_pred, y_test_actual =
evaluate_lstm_model(model, X_test, y_test, scaler,
num_features=scaled_data.shape[1])

    # Return evaluation results
    return lstm_rmse, lstm_mae

# Run the LSTM pipeline and store the results
lstm_rmse, lstm_mae = run_lstm_pipeline('Daily Summaries - GHCND.csv')

print(f'LSTM RMSE: {lstm_rmse:.2f}')
print(f'LSTM MAE: {lstm_mae:.2f}')

def plot_model_comparison(models, rmse_values, mae_values):
    """
    Plots a comparison of RMSE and MAE values across different models.

    Parameters:
    models (list of str): Names of the models.
    rmse_values (list of float): RMSE values for each model.
    mae_values (list of float): MAE values for each model.

    Returns:
```

```
    None
    """
    fig, ax = plt.subplots(figsize=(10, 6))

    bar_width = 0.35
    index = np.arange(len(models))

    bar1 = ax.bar(index, rmse_values, bar_width, label='RMSE',
color='b', alpha=0.7)
    bar2 = ax.bar(index + bar_width, mae_values, bar_width,
label='MAE', color='r', alpha=0.7)

    ax.set_xlabel('Models')
    ax.set_ylabel('Error Value')
    ax.set_title('Comparison of RMSE and MAE across Different Models')
    ax.set_xticks(index + bar_width / 2)
    ax.set_xticklabels(models)
    ax.legend()

    # Display the values on the bars
    for rect in bar1 + bar2:
        height = rect.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

    plt.tight_layout()
    plt.show()

def main_plot_comparison(prophet_rmse, prophet_mae, arima_rmse,
arima_mae, lstm_rmse, lstm_mae):
    """
    Main function to execute the RMSE and MAE comparison plot.

    Parameters:
    prophet_rmse, arima_rmse, lstm_rmse (float): RMSE values for
Prophet, ARIMA, and LSTM models.
    prophet_mae, arima_mae, lstm_mae (float): MAE values for Prophet,
ARIMA, and LSTM models.

    Returns:
    None
    """
    # Model names
models = ['Prophet', 'ARIMA', 'LSTM']

# RMSE and MAE values
rmse_values = [prophet_rmse, arima_rmse, lstm_rmse]
mae_values = [prophet_mae, arima_mae, lstm_mae]

# Plot the comparison
plot_model_comparison(models, rmse_values, mae_values)


def create_sequences(data, seq_length):
    """
```

```
    Creates sequences from time series data for LSTM input.

    Parameters:
    data (np.array): The input data.
    seq_length (int): The length of each sequence.

    Returns:
    tuple: Sequences (X) and corresponding targets (y).
    """
    X = []
    y = []
    # Loop over the dataset to create sequences of the specified length
    for i in range(len(data) - seq_length):
        X.append(data[i:(i + seq_length), :])  # Extract the sequence
of length 'seq_length'
        y.append(data[i + seq_length, 0])  # Target value is the next
value in the sequence (first feature)
    return np.array(X), np.array(y)  # Convert lists to numpy arrays


def time_series_cross_validation(data, seq_length=30, n_splits=5):
    """
    Performs time series cross-validation for LSTM model.

    Parameters:
    data (np.array): The input data.
    seq_length (int): The length of each sequence.
    n_splits (int): The number of cross-validation splits.

    Returns:
    tuple: Average RMSE and MAE scores across all splits.
    """
    tscv = TimeSeriesSplit(n_splits=n_splits)  # Initialize
TimeSeriesSplit object for cross-validation
    rmse_scores = []  # List to store RMSE scores for each fold
    mae_scores = []   # List to store MAE scores for each fold

    # Loop over each fold in the cross-validation
    for train_index, val_index in tscv.split(data):
        train_data, val_data = data[train_index], data[val_index]  #
Split data into training and validation sets
        X_train, y_train = create_sequences(train_data, seq_length)  #
Create sequences for training
        X_val, y_val = create_sequences(val_data, seq_length)        #
Create sequences for validation

        # Define the LSTM model architecture
        model = Sequential()
        model.add(LSTM(50, activation='relu', input_shape=(seq_length,
data.shape[1])))  # LSTM layer with 50 units
        model.add(Dense(1))  # Output layer with a single unit
        model.compile(optimizer='adam', loss='mse')  # Compile the
model with Adam optimizer and MSE loss

        # Early stopping callback to prevent overfitting and save best
model
        early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
```

```
        # Train the LSTM model on the training data with validation on
validation set
        model.fit(X_train, y_train, epochs=20, batch_size=32,
validation_data=(X_val, y_val),
                  verbose=1, callbacks=[early_stopping])

        # Make predictions on the validation set
        y_pred = model.predict(X_val)

        # Calculate RMSE and MAE for the current fold
        rmse = np.sqrt(mean_squared_error(y_val, y_pred))
        mae = mean_absolute_error(y_val, y_pred)

        # Append the calculated scores to the respective lists
        rmse_scores.append(rmse)
        mae_scores.append(mae)

    # Return the average RMSE and MAE scores across all folds
    return np.mean(rmse_scores), np.mean(mae_scores)


def run_lstm_cv_pipeline(filepath, columns, seq_length=30, n_splits=5):
    """
    Runs the LSTM cross-validation pipeline and returns evaluation
metrics.

    Parameters:
    filepath (str): The path to the CSV file.
    columns (list of str): List of columns to include in the model.
    seq_length (int): The length of each sequence.
    n_splits (int): The number of cross-validation splits.

    Returns:
    tuple: Average RMSE and MAE scores across all splits.
    """
    # Load the dataset from the CSV file, parse dates, and set the
'DATE' column as the index
    daily_data = pd.read_csv(filepath, parse_dates=['DATE'],
index_col='DATE')

    # Select the specified columns, convert to numeric values, and drop
any rows with missing values
    data = daily_data[columns].apply(pd.to_numeric,
errors='coerce').dropna().values

    # Perform cross-validation using the LSTM model
    lstm_rmse, lstm_mae = time_series_cross_validation(data,
seq_length=seq_length, n_splits=n_splits)

    # Return the average RMSE and MAE scores
    return lstm_rmse, lstm_mae


# Run the LSTM cross-validation pipeline with the specified file path
and columns
lstm_rmse, lstm_mae = run_lstm_cv_pipeline('Daily Summaries -
GHCND.csv', columns=['TMAX', 'TMIN', 'PRCP', 'SNWD'])
```

```
# Print the results of cross-validation
print(f'LSTM Cross-Validation RMSE: {lstm_rmse:.2f}')
print(f'LSTM Cross-Validation MAE: {lstm_mae:.2f}')


# Define the column names mapping for readability in plots and reports
COLUMN_NAMES = {
    'daily': ['Maximum Temperature', 'Minimum Temperature',
'Precipitation', 'Snow Depth',
              '7-Day Rolling Maximum Temperature', '7-Day Rolling
Minimum Temperature',
              '7-Day Rolling Precipitation', '7-Day Rolling Snow
Depth']
}

def load_and_preprocess_data(filepath):
    """
    Load and preprocess the data.

    Parameters:
    - filepath: str, the path to the CSV file containing the data.

    Returns:
    - scaled_data: np.ndarray, the scaled feature data.
    - scaler: MinMaxScaler object, the scaler used to normalize the
data.
    - columns: list, the names of the columns in the data.
    - daily_data: pd.DataFrame, the original daily data with added
rolling features.
    """
    # Load data from CSV, parse dates, and set the 'DATE' column as the
index
    daily_data = pd.read_csv(filepath, parse_dates=['DATE'],
index_col='DATE')
    daily_data = daily_data.sort_index()  # Ensure the data is sorted
by date

    # Feature engineering: Adding rolling averages for key features (7-
day rolling window)
    daily_data['TMAX_ROLLING'] =
daily_data['TMAX'].rolling(window=7).mean()
    daily_data['TMIN_ROLLING'] =
daily_data['TMIN'].rolling(window=7).mean()
    daily_data['PRCP_ROLLING'] =
daily_data['PRCP'].rolling(window=7).mean()
    daily_data['SNWD_ROLLING'] =
daily_data['SNWD'].rolling(window=7).mean()

    # Define the columns to be used in the model
    columns = ['TMAX', 'TMIN', 'PRCP', 'SNWD', 'TMAX_ROLLING',
'TMIN_ROLLING', 'PRCP_ROLLING', 'SNWD_ROLLING']

    # Convert data to numeric and handle missing values by filling with
the median
    data = daily_data[columns].apply(pd.to_numeric, errors='coerce')
    data = data.fillna(data.median())
```

```
    # Normalize the data using MinMaxScaler to ensure all features are
on a similar scale
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(data)

    return scaled_data, scaler, columns, daily_data

def create_sequences(data, seq_length=30):
    """
    Create sequences of data for LSTM input.

    Parameters:
    - data: np.ndarray, the scaled feature data.
    - seq_length: int, the length of each sequence.

    Returns:
    - X: np.ndarray, the input sequences.
    - y: np.ndarray, the corresponding target values.
    """
    X, y = [], []
    # Loop through the dataset to create sequences of specified length
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])  # Sequence of length
'seq_length'
        y.append(data[i+seq_length])    # Target value is the next
point after the sequence
    return np.array(X), np.array(y)

def build_model(hp, input_shape):
    """
    Build and compile a Bidirectional LSTM model.

    Parameters:
    - hp: HyperParameters object, for tuning the model.
    - input_shape: tuple, the shape of the input data.

    Returns:
    - model: Sequential model, the compiled LSTM model.
    """
    model = Sequential()
    # Add a Bidirectional LSTM layer with tunable units and return
sequences for stacking
    model.add(Bidirectional(LSTM(units=hp.Int('units1', min_value=32,
max_value=256, step=32),
                                 return_sequences=True),
input_shape=input_shape))
    # Add Dropout layer to prevent overfitting with tunable dropout
rate
    model.add(Dropout(hp.Float('dropout1', min_value=0.1,
max_value=0.5, step=0.1)))

    # Add a second LSTM layer with tunable units
    model.add(LSTM(units=hp.Int('units2', min_value=32, max_value=256,
step=32)))

    # Dropout layer for regularization
    model.add(Dropout(hp.Float('dropout2', min_value=0.1,
max_value=0.5, step=0.1)))
```

```
    # Output layer: Dense layer with the number of outputs equal to the
number of features
    model.add(Dense(input_shape[1]))  # Output layer for all features

    # Compile the model with a tunable learning rate

model.compile(optimizer=tf.keras.optimizers.Adam(hp.Choice('learning_ra
te', values=[1e-4, 1e-3, 1e-2])),
                  loss='mean_squared_error')
    return model

def tune_and_train_model(X_train, y_train, input_shape):
    """
    Tune and train the LSTM model using Bayesian Optimization.

    Parameters:
    - X_train: np.ndarray, training input sequences.
    - y_train: np.ndarray, training target values.
    - input_shape: tuple, the shape of the input data.

    Returns:
    - best_model: Sequential model, the best trained LSTM model.
    """
    # Initialize Bayesian Optimization tuner
    tuner = BayesianOptimization(
        lambda hp: build_model(hp, input_shape),  # Pass build_model as
a lambda function
        objective='val_loss',  # Optimization objective
        max_trials=15,  # Number of different hyperparameter
combinations to try
        executions_per_trial=2,  # Number of executions per trial to
reduce variability
        directory='bayesian_tuning',  # Directory to save tuning
results
        project_name='weather_forecast')

    # Define early stopping callback to prevent overfitting and
unnecessary training
    early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

    # Perform hyperparameter search
    tuner.search(X_train, y_train, epochs=20, validation_split=0.2,
callbacks=[early_stopping])

    # Retrieve the best model found during the search
    best_model = tuner.get_best_models(num_models=1)[0]
    return best_model

def predict_future(model, data, scaler, seq_length, num_features,
steps=1095):
    """
    Predict future values using the trained model.

    Parameters:
    - model: Sequential model, the trained LSTM model.
    - data: np.ndarray, the scaled feature data.
```

```
    - scaler: MinMaxScaler object, the scaler used to normalize the
data.
    - seq_length: int, the length of each sequence.
    - num_features: int, the number of features in the data.
    - steps: int, the number of future steps to predict.

    Returns:
    - future_predictions_actual: np.ndarray, the predicted future
values in the original scale.
    """
    # Start with the last sequence of the training data
    last_sequence = data[-seq_length:]
    future_predictions = []

    # Generate predictions for the specified number of future steps
    for _ in range(steps):
        # Predict the next value using the model
        prediction = model.predict(last_sequence.reshape(1, seq_length,
num_features))
        future_predictions.append(prediction[0])
        # Update the sequence with the predicted value
        last_sequence = np.append(last_sequence[1:], prediction,
axis=0)

    # Inverse transform the predictions to their original scale
    future_predictions = np.array(future_predictions)
    future_predictions_actual =
scaler.inverse_transform(future_predictions)
    return future_predictions_actual

def calculate_metrics(y_test, y_pred, columns):
    """
    Calculate and print RMSE and MAE metrics for the test set.

    Parameters:
    - y_test: np.ndarray, the actual test set values.
    - y_pred: np.ndarray, the predicted test set values.
    - columns: list, the names of the columns being predicted.
    """
    # Calculate RMSE and MAE for each feature in the test set
    rmse = np.sqrt(mean_squared_error(y_test, y_pred,
multioutput='raw_values'))
    mae = mean_absolute_error(y_test, y_pred, multioutput='raw_values')

    # Create a DataFrame to store and display the metrics for each
feature
    metrics_df = pd.DataFrame({
        'Parameter': columns,
        'RMSE': rmse,
        'MAE': mae
    })

    # Print the metrics
    print(metrics_df)

def plot_predictions(daily_data, future_predictions, future_dates,
columns, ci_multiplier=1.96):
    """
```

```
    Plot actual vs forecasted data with confidence intervals.

    Parameters:
    - daily_data: pd.DataFrame, the actual historical data.
    - future_predictions: np.ndarray, the predicted future values.
    - future_dates: pd.DatetimeIndex, the future dates corresponding to
the predictions.
    - columns: list, the names of the columns being predicted.
    - ci_multiplier: float, the multiplier for the confidence interval
(default is 1.96 for 95% CI).
    """
    # Filter out rolling columns since they are derived features
    non_rolling_columns = [col for col in columns if 'ROLLING' not in
col]

    for i, col in enumerate(non_rolling_columns):
        plt.figure(figsize=(10, 6))

        # Use COLUMN_NAMES to get full names for the columns
        full_col_name = COLUMN_NAMES['daily'][columns.index(col)]
        print("Plotting column:", full_col_name)  # Debugging statement

        # Plot actual historical data
        plt.plot(daily_data.index, daily_data[col], label='Actual ' +
full_col_name, color='blue')

        # Plot forecasted data
        plt.plot(future_dates, future_predictions[:, i],
label='Forecasted ' + full_col_name, linestyle='dashed',
color='orange')

        # Calculate standard deviation of residuals for confidence
interval
        residuals = daily_data[col][-len(future_predictions):] -
future_predictions[:, i]
        residual_std = residuals.std()

        # Calculate the confidence interval bounds
        lower_bound = future_predictions[:, i] - ci_multiplier *
residual_std
        upper_bound = future_predictions[:, i] + ci_multiplier *
residual_std

        # Plot the confidence interval
        plt.fill_between(future_dates, lower_bound, upper_bound,
color='orange', alpha=0.2, label='95% CI')

        plt.title(f'Forecast for {full_col_name}')
        plt.xlabel('Date')
        plt.ylabel(full_col_name)
        plt.legend()
        plt.grid(True)
        plt.show()

def main():
    """
    Main function to load data, train the LSTM model, predict future
values, and plot results.
```

```
    """
    # Filepath for the dataset
    filepath = 'Daily Summaries - GHCND.csv'
    seq_length = 30  # Length of sequences for LSTM input

    # Load and preprocess data
    scaled_data, scaler, columns, daily_data =
load_and_preprocess_data(filepath)

    # Create input sequences and corresponding target values for LSTM
    X, y = create_sequences(scaled_data, seq_length)

    # Split data into train and test sets (80-20 split)
    train_size = int(len(X) * 0.8)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]

    # Define input shape for the model
    input_shape = (X_train.shape[1], X_train.shape[2])

    # Tune hyperparameters and train the LSTM model
    model = tune_and_train_model(X_train, y_train, input_shape)

    # Make predictions on the test set and inverse transform to
original scale
    y_pred_scaled = model.predict(X_test)
    y_pred_actual = scaler.inverse_transform(y_pred_scaled)
    y_test_actual = scaler.inverse_transform(y_test)

    # Calculate RMSE and MAE for the test set predictions
    calculate_metrics(y_test_actual, y_pred_actual, columns)

    # Predict the next 3 years (1095 days) using the trained model
    future_predictions = predict_future(model, scaled_data, scaler,
seq_length, num_features=scaled_data.shape[1], steps=1095)

    # Generate future dates for plotting
    future_dates = pd.date_range(start=daily_data.index[-1],
periods=1095, freq='D')

    print("Columns used for predictions:", columns)

    # Plot actual and forecasted data with confidence intervals
    plot_predictions(daily_data, future_predictions, future_dates,
columns)

if __name__ == "__main__":
    main()
```