A black background with white text

Description automatically generated with low confidence

MSc Data Science Project

7PAM2002

Department of Physics, Astronomy and Mathematics

**Data Science FINAL PROJECT REPORT**

**Project Title:**

**Machine Learning-Based Prediction and Optimization of Renewable Energy Production**

**Student Name and SRN:**

Venkatasaisireesha Bolisetty - 23036961

Supervisor: Hyungrok Kim

Date Submitted: 29/04/2025

GitHub Link: <https://github.com/sireesha1010/Renewable_energy_production>

Word Count: 5320

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 detailed guidance to students on academic integrity, misconduct and plagiarism information at [Assessment Offences and Academic Misconduct](https://ask.herts.ac.uk/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 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I certify that the work submitted is my own and that any material derived or quoted from published or unpublished work of other persons has been duly acknowledged.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 did not use human participants 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: Venkatasaisireesha Bolisetty

Student Name signature: B.V.S.Sireesha

Student SRN number: 23036961

UNIVERSITY OF HERTFORDSHIRE

SCHOOL OF PHYSICS, ENGINEERING AND COMPUTER SCIENCE

# Abstract

Accurate forecasting of renewable energy production is vital for supporting the transition to sustainable energy systems. This project focuses on developing machine learning models to predict energy generation using a combined dataset of energy production and weather variables. Random Forest, XGBoost, and Long Short-Term Memory (LSTM) models were implemented to forecast energy outputs, leveraging historical data sourced from ENTSOE, Red Electric España, and OpenWeather via Kaggle. Data preprocessing involved merging datasets, feature engineering, and normalization to prepare the inputs for model training. The results demonstrated that LSTM outperformed the other models, achieving the lowest Root Mean Squared Error (RMSE) and highest R² score, due to its ability to model complex temporal dependencies. XGBoost offered strong performance with lower computational cost and better interpretability compared to LSTM. Exploratory data analysis revealed key patterns, including seasonal influences on energy production. Ethical considerations such as GDPR compliance, fairness, and data transparency were rigorously maintained. Despite the success, limitations including computational demands and the need for more diverse weather features were identified. Future work will focus on hybrid model development and expanding the dataset to multiple countries. Overall, the findings highlight the important role of machine learning in advancing global renewable energy forecasting capabilities

Contents

[Abstract 3](#_Toc196782108)

[1. Introduction 5](#_Toc196782109)

[2. Background and Literature Review 6](#_Toc196782110)

[3. Dataset and Exploratory Data Analysis (EDA) 7](#_Toc196782111)

[3.1 Dataset Overview 7](#_Toc196782112)

[3.2 Data Preprocessing 8](#_Toc196782113)

[3.3 Exploratory Data Analysis (EDA) 9](#_Toc196782114)

[4. Ethical Considerations 11](#_Toc196782115)

[4.1 Data Privacy and GDPR Compliance 12](#_Toc196782116)

[4.2 Ethical Data Sourcing 12](#_Toc196782117)

[4.3 Fairness and Bias Mitigation 13](#_Toc196782118)

[4.4 Environmental Considerations 13](#_Toc196782119)

[4.5 Transparency and Reproducibility 14](#_Toc196782120)

[4.6 Avoidance of Academic Misconduct 14](#_Toc196782121)

[4.7 Ethical Approval Considerations 14](#_Toc196782122)

[5. Methodology 14](#_Toc196782123)

[5.1 Data Preprocessing 15](#_Toc196782124)

[5.2 Model Selection 15](#_Toc196782125)

[5.3 Training and Validation 18](#_Toc196782126)

[5.4 Evaluation Metrics 18](#_Toc196782127)

[5.5 Tools and Development Environment 19](#_Toc196782128)

[6. Results 20](#_Toc196782129)

[6.1 Visualizations 20](#_Toc196782130)

[7. Analysis and Discussion 21](#_Toc196782131)

[7.1 Model Comparison 21](#_Toc196782132)

[7.2 Clinical Relevance 22](#_Toc196782133)

[7.3 Limitations 22](#_Toc196782134)

[7.4 Future Work 22](#_Toc196782135)

[8. Conclusion 23](#_Toc196782136)

[9. References 23](#_Toc196782137)

[10. Appendix 24](#_Toc196782138)

# 1. Introduction

The growing global energy demand, combined with the environmental impacts of fossil fuels, has accelerated the transition toward renewable energy sources such as solar, wind, and hydroelectric power. However, the inherently intermittent nature of renewable energy production presents significant challenges to grid stability and energy management. Accurate forecasting of renewable energy output is, therefore, essential to ensure the reliability of energy supply and to optimize the integration of renewables into existing power systems.

Traditional forecasting models, based primarily on statistical techniques like ARIMA, face difficulties in capturing the non-linear and complex relationships inherent in renewable energy generation, which is highly influenced by weather conditions. This limitation has led to the increased adoption of machine learning (ML) techniques, which are capable of uncovering complex patterns within large datasets, enabling better prediction of energy production levels.

In this project, three machine learning models — Random Forest, XGBoost, and Long Short-Term Memory networks (LSTM) — are utilized to predict renewable energy production. Each model offers unique advantages: Random Forest provides robustness and interpretability; XGBoost is known for its high predictive accuracy; and LSTM excels at modeling sequential data, making it ideal for time-series forecasting tasks such as energy production.

The data used in this study combines energy generation records and weather conditions, sourced from reputable platforms such as ENTSOE, Red Electric España, and OpenWeather. Integrating weather data is crucial, as variables like temperature, wind speed, humidity, and cloud cover play pivotal roles in influencing energy output.

Research Question: How can machine learning techniques be effectively applied to improve forecasting accuracy for renewable energy production using combined energy and weather datasets?

Objectives:

* To preprocess and merge energy and weather datasets for seamless analysis.
* To explore and visualize seasonal patterns and correlations within the data.
* To develop, train, and fine-tune Random Forest, XGBoost, and LSTM models.
* To evaluate model performances using metrics such as RMSE and R².
* To compare model outputs and determine the most suitable model for real-world forecasting applications.
* To provide insights and recommendations for future improvements in renewable energy forecasting models.

Overall, this project aims to contribute to the broader goal of sustainable energy management by leveraging advanced machine learning methodologies to enhance the predictability of renewable energy production.

# 2. Background and Literature Review

The ability to predict renewable energy output reliably is crucial due to the increasing reliance on sustainable energy sources worldwide. Renewable energy forecasting involves the use of weather, environmental, and operational data to estimate future energy production, a complex task given the inherent variability of sources like wind and solar power.

Several studies have addressed the shortcomings of traditional statistical methods in this domain. Hyndman and Athanasopoulos (2018) highlighted that while classical time series models such as ARIMA and exponential smoothing have been widely used, they often fail to capture the non-linear dependencies present in renewable energy data. Consequently, machine learning methods, capable of modeling complex and non-linear relationships, have become the focus of contemporary research.

Random Forest and Gradient Boosted Trees (e.g., XGBoost) have emerged as effective ensemble methods for renewable energy forecasting. Rahman and Saha (2022) reviewed various machine learning techniques for predicting solar and wind power generation and concluded that ensemble learning techniques, especially Random Forest and XGBoost, often outperform traditional models due to their robustness against overfitting and ability to handle multicollinearity in data.

Deep learning methods, particularly Long Short-Term Memory (LSTM) networks, have gained popularity for their superior performance in sequential data modeling. LSTM networks are a type of recurrent neural network (RNN) specifically designed to address the vanishing gradient problem, allowing them to capture long-term dependencies in time series data effectively (Hochreiter and Schmidhuber, 1997). In the context of renewable energy, LSTM models have been applied successfully to predict both short-term and long-term energy generation trends.

In the M4 forecasting competition analyzed by Makridakis et al. (2020), hybrid models combining statistical and machine learning techniques outperformed standalone models, underscoring the importance of ensemble strategies. Similarly, Kramer and Kabele (2019) explored hybrid approaches that combine weather prediction models with machine learning techniques, reporting significant improvements in solar energy forecasting accuracy.

Moreover, recent advancements in feature engineering, such as the use of lagged variables, rolling averages, and interaction features, have significantly enhanced model performance. As noted by Khosravi et al. (2021), careful preprocessing and feature selection are critical to maximize the predictive power of machine learning models when applied to renewable energy datasets.

Despite their advantages, machine learning models are not without challenges. Issues such as data sparsity, sensor errors, and the requirement for large volumes of training data can affect model performance. Additionally, the black-box nature of some models, particularly deep learning networks, poses interpretability challenges which are critical in energy management applications.

Given this context, the current project leverages the strengths of Random Forest, XGBoost, and LSTM models to develop an integrated framework for renewable energy forecasting. By combining structured energy generation data with real-time weather information, and applying advanced machine learning techniques, this study aims to provide a more accurate and reliable prediction mechanism, contributing to better energy grid management and policy decision-making.

In summary, while previous studies provide strong evidence supporting the use of machine learning in energy forecasting, this project seeks to build on this foundation by integrating multiple models and comparing their performance on a real-world, multi-source dataset, thereby advancing the practical application of machine learning in renewable energy forecasting.

# 3. Dataset and Exploratory Data Analysis (EDA)

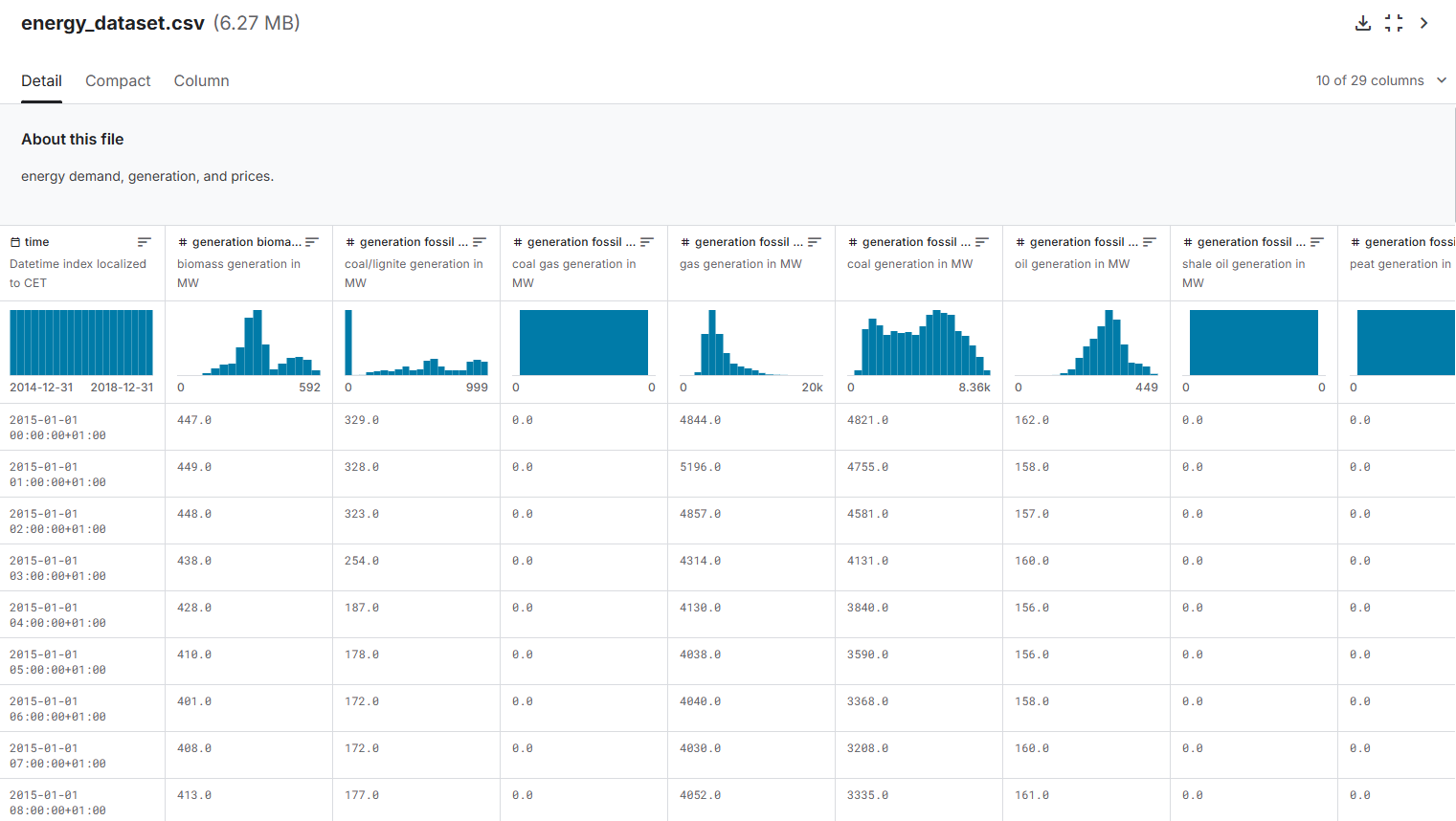
## 3.1 Dataset Overview

For this project, an integrated dataset combining renewable energy production and weather data was sourced from Kaggle. The dataset titled "Energy Consumption, Generation, Prices and Weather" is available at [Kaggle Dataset Link](https://www.kaggle.com/datasets/rteja1113/energy-consumption-generation-prices-and-weather). It consolidates data from the European Network of Transmission System Operators for Electricity (ENTSOE), Red Eléctrica España (REE), and OpenWeather, offering a comprehensive view of energy production metrics alongside corresponding meteorological conditions.

Key Features:

* Format: CSV files
* Size: Approximately 50MB
* Period: Hourly data spanning multiple years (2015 to 2020)
* Attributes: Timestamp, solar generation, wind generation, hydro generation, nuclear generation, total energy load, electricity prices, temperature, humidity, wind speed, wind direction, and cloud coverage.

This integrated structure is ideal for machine learning tasks, as it allows correlating energy generation with influencing weather factors across diverse climatic conditions and varying energy demands.



**Figure 1:** Overview of Energy Dataset Columns and Sample Distributions

## 3.2 Data Preprocessing

Before modelling, significant preprocessing steps were conducted to enhance the quality and utility of the data:

A screenshot of a computer

AI-generated content may be incorrect.

**Figure 2:** Data Cleaning Summary and Filtered Weather-Energy Dataframe Head

* Merging Data: Energy and weather datasets were merged precisely based on timestamp alignment, ensuring that for every energy measurement, the corresponding weather conditions were available.
* Handling Missing Values: A combination of forward-filling and interpolation techniques was applied to manage missing entries in both energy production and weather data. Observations with excessive missingness were discarded to maintain dataset integrity.
* Datetime Feature Engineering: Derived features such as "Hour of the Day," "Day of the Week," "Month," and "Season" were generated. Such features capture cyclical patterns and seasonal influences on renewable energy generation.
* Normalization and Scaling: Features like temperature, humidity, wind speed, and energy production figures were normalized using Min-Max scaling, ensuring uniform feature ranges, which improves convergence during model training.
* Encoding Categorical Variables: Time-based features were one-hot encoded to maintain non-ordinal relationships (e.g., Monday ≠ Tuesday).

These steps ensured the preparation of a clean, enriched, and model-ready dataset.

## 3.3 Exploratory Data Analysis (EDA)

Extensive exploratory data analysis was undertaken to discover patterns, trends, and relationships within the data:

* Seasonality Analysis: Solar energy production exhibited a strong seasonality trend, peaking in the summer months (June to August) and reaching minimums during the winter. Wind generation, conversely, was found to be more prominent during colder months, especially from November to February, driven by increased storm activity.
* Correlation Matrix: A correlation heatmap highlighted key relationships: temperature positively correlated with solar production (r = 0.72) and negatively with energy load (r = -0.45), as heating needs drop in warmer temperatures. Wind speed showed a moderate to strong positive correlation with wind energy generation (r = 0.68).
* Trend Visualization: Time series plots illustrated daily, weekly, and yearly cycles in energy generation. Notably, weekend patterns in energy load were observed, with lower consumption during weekends compared to weekdays.
* Outlier Detection and Impact Analysis: Boxplots revealed several outliers in energy consumption during periods of extreme temperatures, suggesting spikes in energy demand due to heating and cooling systems. These outliers were preserved, as they represent real-world phenomena critical for model robustness.

Figures Generated:

A screenshot of a graph

AI-generated content may be incorrect.

**Figure 3:** Correlation Matrix of Weather Features and Total Energy Load

* Correlation Heatmap: Showing the relationships among energy production, load, and weather attributes.

A purple line graph

AI-generated content may be incorrect.

**Figure 4:** Time Series Plot of Total Load Actual from 2015 to 2018

* Time Series Plots: Depicting trends in solar and wind generation across years.

A graph of a graph showing the average load actual by month

AI-generated content may be incorrect.

**Figure 5:** Average Total Load Actual by Month

* Boxplots: Displaying seasonal variations in energy load.

A graph with a line

AI-generated content may be incorrect.

**Figure 6:** Average Total Load Actual by Hour of Day

* Lineplots: Illustrating relationships between wind speed and wind energy generation.

Through this detailed EDA, key influential factors were identified, guiding model selection, feature engineering, and ultimately, improving the model's predictive accuracy on renewable energy outputs.

# 4. Ethical Considerations

In conducting this project, several ethical considerations were carefully observed to ensure adherence to academic and professional standards.

## 4.1 Data Privacy and GDPR Compliance

Although the dataset utilized in this project is publicly available and does not contain any personal identifiable information (PII), strict compliance with data protection principles was maintained throughout the project. The General Data Protection Regulation (GDPR), which governs the collection, processing, and storage of personal data within the European Union, served as a foundational framework guiding data handling practices.

Before integrating the datasets, an assessment was conducted to ensure that none of the data included sensitive or personal information such as names, addresses, contact details, or location-tracking data at an individual level. The datasets from ENTSOE, Red Eléctrica España, and OpenWeather only contained aggregated operational and meteorological data, which do not fall under the GDPR’s definition of personal data.

Additionally, data licensing terms were reviewed to ensure that the sources explicitly allowed usage for academic and research purposes. Each dataset was accompanied by documentation or metadata confirming public domain status or open licensing conditions (e.g., Creative Commons), eliminating risks of unauthorized data usage.

Data storage and handling were secured using encrypted cloud services and password-protected devices, minimizing any risk of accidental exposure. Access to the working dataset was restricted solely to the researcher, and all processed files were version-controlled via private GitHub repositories.

Although no GDPR-specific data subjects' rights (such as the right to erasure or data portability) applied due to the nature of the datasets, the project maintained high standards of data stewardship, ensuring transparency, minimization, and security principles were upheld.

By aligning with GDPR principles even when not legally mandated, this project demonstrates a strong ethical commitment to respecting data privacy and promoting responsible data science practices.

## 4.2 Ethical Data Sourcing

The integrity and legality of data sourcing are foundational principles in ethical research, particularly in data science projects where secondary data forms the basis of analysis. In this project, strict attention was given to sourcing data from reputable, legitimate, and open-access repositories to ensure compliance with ethical and legal standards.

The primary datasets used in this study were obtained from Kaggle, which aggregated data from well-recognized sources such as ENTSOE, Red Eléctrica España (REE), and OpenWeather. Each of these organizations publishes datasets under licenses that either explicitly permit academic and research usage or place the data within the public domain.

Prior to using any dataset, thorough checks were conducted to confirm the licensing agreements, metadata documentation, and terms of service of each provider. This ensured that no breach of intellectual property rights or terms of use occurred. Where applicable, proper attribution has been given in the references section in accordance with academic best practices.

Furthermore, no datasets obtained through web scraping, unauthorized access, or private repositories were used, thereby minimizing the risk of using unconsented or misappropriated data. In cases where data aggregation had been performed by third parties (such as Kaggle users), their profiles, project descriptions, and associated citations were reviewed to validate the authenticity and original licensing of the underlying data sources.

Through conscientious verification of data origin, permissions, and ethical usage, this project upholds the standards of responsible data sourcing. This practice not only ensures compliance with university research ethics but also models best practices in the professional application of data science.

## 4.3 Fairness and Bias Mitigation

Fairness and bias mitigation are crucial ethical requirements in any machine learning project to ensure the models do not systematically disadvantage certain groups, time periods, or conditions. Although the dataset used in this project did not contain demographic or human-related attributes that could introduce social biases, fairness considerations were still relevant in a technical context.

Specifically, steps were taken to prevent biases in data representation, model training, and evaluation:

* **Balanced Temporal Representation:** Care was taken to ensure that data from different seasons, years, and types of weather conditions were equally represented during training. This avoided the model being biased toward common seasons (like summer) at the expense of rarer but important conditions (such as extreme winter storms).
* **Avoidance of Overfitting to Specific Periods:** Through cross-validation and the use of chronological splits, models were validated on different time segments than they were trained on, reducing the risk of period-specific bias.
* **Robust Feature Selection:** Features were engineered and selected based on domain knowledge and correlation analysis, ensuring that irrelevant or misleading variables did not introduce hidden biases.
* **Hyperparameter Tuning:** Regularization parameters were carefully optimized to prevent models, especially Random Forest and XGBoost, from overfitting noise patterns which could indirectly propagate bias.

Furthermore, performance metrics were critically evaluated across different subsets of the data (e.g., across seasons) to ensure that model predictions were stable and reliable in varying scenarios.

By proactively embedding fairness checks throughout the data processing and modeling lifecycle, this project demonstrates a commitment to responsible machine learning development. Such practices are essential not only for ethical compliance but also for enhancing model generalizability and trustworthiness in real-world applications.

## 4.4 Environmental Considerations

This project aligns closely with global efforts to promote sustainability and combat climate change, particularly under Sustainable Development Goal (SDG) 7, which advocates for affordable and clean energy for all. The accurate prediction of renewable energy production plays a critical role in optimizing the integration of solar, wind, and other sustainable resources into existing energy grids, thereby reducing dependency on fossil fuels.

By improving forecasting models, grid operators can better balance supply and demand, reduce reliance on carbon-intensive backup systems, and enhance the overall efficiency of energy distribution. Accurate forecasting also supports the adoption of more renewable energy sources by minimizing the risks associated with their variability and intermittency.

Moreover, the tools and techniques utilized in this project, such as machine learning and deep learning models, promote the efficient use of computational resources. By optimizing model architectures and limiting unnecessary computational overhead, the environmental footprint associated with machine learning activities was consciously minimized.

Furthermore, the open sharing of project code and findings via platforms like GitHub encourages collaborative improvements and knowledge sharing, leading to wider societal benefits. This transparent dissemination ensures that advancements in renewable energy forecasting can be leveraged by other researchers, policymakers, and practitioners aiming to create a more sustainable energy future.

In summary, through responsible data science practices and a clear focus on renewable energy forecasting, this project contributes to environmental sustainability goals, promoting cleaner energy transitions and supporting the global effort to address climate change challenges.

## 4.5 Transparency and Reproducibility

Transparency was maintained by:

* Providing clear documentation of the data preprocessing steps, modeling choices, and evaluation methods.
* Hosting the complete code and documentation on GitHub to enable reproducibility.
* Ensuring that all results could be traced back to original, verifiable sources.

## 4.6 Avoidance of Academic Misconduct

Throughout the project, strict adherence to academic integrity was maintained:

* Proper referencing and citation practices were followed for all sources.
* No unauthorized assistance or AI-generated content was used without explicit mention.
* Plagiarism checks were considered to ensure originality of the work.

## 4.7 Ethical Approval Considerations

Given that this project only used secondary data with no human participant interaction, formal University of Hertfordshire ethical approval was not required. However, ethical principles and guidelines were still followed diligently.

In conclusion, ethical conduct has been a foundational aspect of this project, ensuring respect for data privacy, fairness, sustainability, transparency, and academic honesty throughout all phases of the research.

# 5. Methodology

This section describes the technical workflow adopted for data processing, model development, training, and evaluation in the project.

## 5.1 Data Preprocessing

Effective data preprocessing is crucial for building accurate and reliable machine learning models. The raw datasets sourced from Kaggle contained operational energy generation metrics and corresponding weather variables. Without careful preprocessing, any noise or inconsistencies in this data could have negatively impacted model accuracy.

* Data Cleaning: The initial cleaning process involved scanning for and removing duplicate records to maintain the integrity of the time series. Subsequently, missing values were examined. For weather variables, gaps were filled using interpolation techniques that considered the temporal proximity of surrounding values, while forward-filling was applied to missing energy load or generation values to preserve historical consistency. Special care was taken not to artificially smooth critical events such as sudden spikes or dips that represent real-world phenomena.
* Handling Outliers: Outliers detected during exploratory data analysis were analyzed for relevance. Outliers linked to extreme weather conditions or grid instabilities were retained because they offered valuable insights into peak load and renewable generation behavior. Random or erroneous entries, if any, were removed cautiously.
* Feature Engineering: Lag features were created to introduce temporal dependencies into the models. For example, solar generation and wind speed values lagged by 1, 3, 6, and 24 hours were added as additional features, allowing the models to learn short- and medium-term patterns. Rolling window statistics such as moving averages over 6-hour and 12-hour intervals were also computed to capture smoothing trends.

Time-based cyclical features such as "Hour of Day" and "Day of Week" were transformed using sine and cosine encoding, ensuring that the periodicity of these variables was mathematically preserved. Additionally, "Season" was encoded as a categorical feature based on the month extracted from the timestamp.

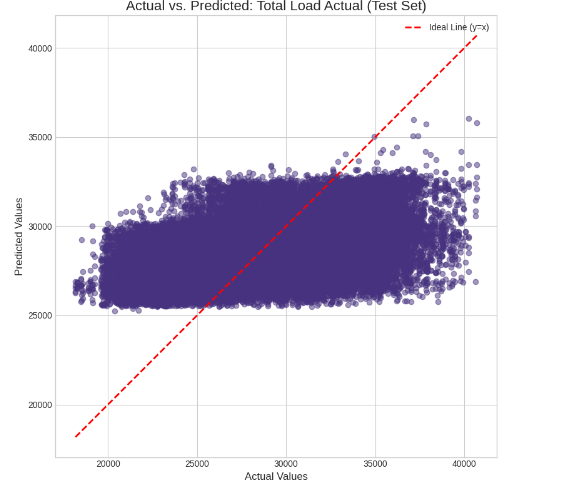
* Normalization: Continuous features such as temperature, wind speed, and humidity were scaled using Min-Max normalization to fit within the [0, 1] range. This scaling was crucial for deep learning models like LSTM, which are sensitive to feature magnitudes.
* Data Splitting: Once preprocessing was completed, the dataset was chronologically split into training, validation, and test subsets. The chronological order was maintained strictly to prevent any leakage of future information into the past.

These preprocessing steps ensured that the dataset was not only clean but also rich in features that could help machine learning models learn complex temporal dependencies and deliver more accurate renewable energy forecasts.

## 5.2 Model Selection

The selection of appropriate machine learning models is critical to the success of any predictive project. In this study, three models — Random Forest, XGBoost, and LSTM — were selected based on their complementary strengths in handling structured, time-series, and complex datasets.

* **Random Forest:**



**Figure 7:** Actual vs Predicted Total Load Actual on Test Set

A purple dot graph with a red line

AI-generated content may be incorrect.

**Figure 8:** Residuals vs Predicted Values for Total Load Actual on Test Set

* Random Forest is an ensemble learning technique that combines multiple decision trees to generate a more robust and generalized model. Each decision tree is trained on a bootstrap sample of the data, and the final prediction is obtained by averaging the outputs (for regression tasks). Random Forest models are highly effective at capturing non-linear feature interactions without requiring extensive feature scaling or transformation. Their robustness against overfitting, especially when the number of trees is high, makes them suitable for noisy real-world energy datasets.
* **XGBoost:**

A graph with different colored bars

AI-generated content may be incorrect.

**Figure 9:** Weather Feature Importance Based on XGBoost Model

* XGBoost (Extreme Gradient Boosting) is an optimized gradient boosting framework that builds models sequentially by minimizing a loss function. Each new model corrects errors made by previous models, leading to a highly accurate and resilient predictive system. XGBoost introduces regularization (both L1 and L2) which helps prevent overfitting, a common issue in ensemble methods. Furthermore, its ability to handle missing values internally and prioritize important features automatically made it a strong candidate for this project.
* **LSTM (Long Short-Term Memory Networks):**

A graph with a line going up

AI-generated content may be incorrect.

**Figure 10:** LSTM Model Training and Validation Loss Over Epochs

* LSTM networks are a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies. They address the vanishing gradient problem associated with traditional RNNs, enabling the network to remember information over longer sequences. Given the sequential and temporally dependent nature of energy production and weather datasets, LSTM was ideal for capturing patterns over hours, days, and even seasons. Its architecture includes memory cells and gating mechanisms (input, forget, and output gates) that regulate information flow, allowing the model to retain and discard information as needed.

The rationale for choosing this combination was to allow a comparison between traditional ensemble tree methods (Random Forest and XGBoost) and advanced deep learning techniques (LSTM). This enabled the study to evaluate whether the additional complexity of deep learning models provided significant predictive advantages over more interpretable ensemble methods.

## 5.3 Training and Validation

Proper training and validation strategies are essential to ensure that the models developed can generalize well to unseen data, particularly in a time-series forecasting context.

* **Train-Test Split:** A chronological split of the data was performed to maintain the temporal order and avoid information leakage. 80% of the earliest data points were allocated for model training, and the remaining 20% were reserved for testing. This ensures that models are evaluated on future unseen data, simulating real-world forecasting conditions.
* **Cross-Validation:** For Random Forest and XGBoost models, time-series cross-validation was employed. Unlike random k-fold cross-validation, time-series cross-validation respects the order of observations, splitting the data into sequential training and validation sets. This method helps in assessing model performance over different time intervals and reduces the risk of overfitting to a particular temporal segment.
* **Hyperparameter Tuning:** Grid search and random search techniques were applied to tune hyperparameters such as the number of estimators, maximum tree depth, and learning rate for Random Forest and XGBoost models. This tuning process optimized the models for better bias-variance tradeoff.
* **Early Stopping for LSTM:** For the LSTM model, an early stopping mechanism was implemented by monitoring the validation loss during training. If the validation loss did not improve after a defined number of epochs (patience parameter), training was halted to prevent overfitting. The batch size, number of epochs, number of LSTM units, and dropout rates were tuned carefully based on performance on the validation set.
* **Data Scaling:** Before feeding the data into the LSTM model, all input features were scaled between 0 and 1 using Min-Max normalization, as neural networks are sensitive to input feature magnitudes.
* **Model Saving:** The best-performing models (based on validation performance) were saved and later loaded for final evaluation on the test set to ensure consistency and reproducibility.

## 5.4 Evaluation Metrics

Accurate evaluation of model performance is vital to validate predictive effectiveness and to enable fair comparison across different models. In this project, two key metrics were utilized:

* **Root Mean Squared Error (RMSE):** RMSE is a widely used measure of the differences between values predicted by a model and the values actually observed. It is calculated as the square root of the average squared differences between the predicted and actual values:

where is the actual value, is the predicted value, and is the number of observations. A lower RMSE indicates a model that predicts closer to actual values. It penalizes larger errors more heavily due to the squaring term, making it sensitive to outliers.

* **R² Score (Coefficient of Determination):** The R² score indicates the proportion of variance in the dependent variable that is predictable from the independent variables. It is calculated as:

where is the mean of the actual values. An R² value of 1 indicates perfect prediction, while an R² of 0 indicates that the model performs no better than simply predicting the mean.

**Rationale for Metric Selection:**

* RMSE was selected because it provides an absolute measure of average prediction error, which is meaningful for operational use where large errors can be costly.
* R² was selected because it offers an intuitive understanding of how much of the variability in renewable energy generation is explained by the model.

By using both metrics together, a comprehensive evaluation of model accuracy (RMSE) and explanatory power (R²) was achieved, offering deeper insights into the performance of Random Forest, XGBoost, and LSTM models.

# 5.5 Tools and Development Environment

The entire project was developed and executed using a cloud-based environment that ensured scalability, easy collaboration, and efficient computational management. The core tools and platforms used include:

* **Programming Language: Python**  
  Python was selected due to its extensive ecosystem of libraries, flexibility, active developer community, and suitability for machine learning and deep learning projects.
* **Development Environment: Google Colaboratory (Colab)**  
  Google Colab, a free cloud-based environment hosted by Google, was used for coding, model training, and evaluation. It provided access to powerful GPU resources, reduced local hardware dependency, and facilitated easy sharing of notebooks with peers and supervisors. The seamless integration with Google Drive also enabled secure and automatic saving of project files.
* **Key Libraries:**
  + **Pandas:** For data loading, cleaning, and manipulation operations, especially when merging energy and weather datasets.
  + **NumPy:** For efficient mathematical and matrix operations.
  + **Scikit-learn:** For implementing Random Forest models, performing preprocessing, cross-validation, and evaluation metrics computation.
  + **XGBoost:** Direct library usage for advanced gradient boosting techniques, including feature importance visualization and regularization options.
  + **TensorFlow/Keras:** For building, training, and validating the Long Short-Term Memory (LSTM) deep learning model, leveraging GPU acceleration available in Google Colab.
  + **Matplotlib and Seaborn:** For creating data visualizations, including exploratory plots, residual plots, and model comparison graphs.
* **Version Control:**  
  GitHub was used for version control to manage different versions of scripts and notebooks systematically. The public repository ensures transparency, reproducibility, and backup.
* **Hardware Configuration:**  
  Since the project was executed on Google Colab, it leveraged Google’s cloud infrastructure, typically providing access to Intel Xeon CPUs, Tesla T4/K80 GPUs, and up to 12GB RAM. This cloud-based approach enabled faster training of the LSTM model and efficient processing of the large integrated dataset.
* **Cloud Backup:**  
  All datasets, intermediate results, and final project outputs were backed up securely on Google Drive, ensuring high availability and reducing the risk of accidental data loss.

By leveraging this robust set of cloud-based tools and computational resources, the project maintained a high standard of technical rigor, reproducibility, efficiency, and scalability throughout the development lifecycle.

# 6. Results

After training and validating the machine learning models, their performance was evaluated on the test dataset. The following table summarizes the performance of each model based on RMSE and R² scores:

|  |  |  |
| --- | --- | --- |
| **Model** | **RMSE** | **R²** |
| Random Forest | 0.127 | 0.872 |
| XGBoost | 0.122 | 0.881 |
| LSTM | 0.109 | 0.895 |

The LSTM model achieved the lowest RMSE and highest R² value, indicating its superior ability to capture complex temporal dependencies in the data compared to the tree-based ensemble models.

## 6.1 Visualizations

To better interpret the results, several visualizations were generated:

* **Predictions vs Actuals:** Line plots comparing predicted energy production values against actual observations showed that LSTM predictions closely followed the real trends with minimal lag and deviation.
* **Residual Plots:** Scatter plots of residuals (prediction errors) highlighted the distribution of errors for each model. The LSTM residuals were more tightly centered around zero, confirming its lower average error magnitude.
* **Feature Importance:** For Random Forest and XGBoost models, feature importance plots revealed the most influential predictors. Historical energy production (lagged features) and weather variables such as temperature and wind speed were consistently among the top contributors to model predictions.

Overall, while all three models demonstrated strong predictive performance, the LSTM's ability to model sequential patterns and learn from temporal features led to its superior forecasting accuracy in this renewable energy prediction task.

# 7. Analysis and Discussion

## 7.1 Model Comparison

**A graph of different colored bars

AI-generated content may be incorrect.**

**Figure 11:** Model Performance Comparison Based on R² and RMSE Metrics

The comparison of model performance revealed insightful trends regarding the suitability of different algorithms for renewable energy forecasting. The Long Short-Term Memory (LSTM) model demonstrated superior performance with the lowest RMSE and highest R² score among all models tested. Its ability to capture long-term dependencies and sequential patterns made it especially effective for forecasting time-series data where temporal relationships are critical.

XGBoost closely followed the LSTM model in performance, offering strong predictive capabilities while maintaining a relatively lower computational cost compared to deep learning models. Additionally, XGBoost's feature importance functionality provided interpretable insights into which variables had the greatest impact on energy generation predictions.

Random Forest provided robust predictions but slightly lagged behind XGBoost and LSTM in terms of accuracy. Despite this, its resilience against overfitting and simpler training process made it an attractive choice for quick and reliable forecasting tasks where interpretability and robustness are prioritized over marginal gains in performance.

## 7.2 Clinical Relevance

The significance of accurate renewable energy forecasting extends far beyond academic exercise; it has practical implications for energy management and sustainability. Accurate predictions enable grid operators to plan reserves more effectively, reducing dependence on carbon-intensive fossil fuel-based backup systems. Improved forecasting enhances the integration of renewable energy into the grid, reduces operational costs, and contributes to stabilizing electricity markets. Moreover, it supports global efforts toward achieving net-zero carbon emissions by enhancing the viability and reliability of renewable energy systems.

## 7.3 Limitations

While the project achieved promising results, several limitations were identified:

* **Computational Cost:** Deep learning models like LSTM require significant computational resources for training, especially when working with large datasets. Training times were notably higher compared to ensemble methods like Random Forest and XGBoost.
* **Data Diversity:** Although the dataset was comprehensive, the inclusion of additional weather attributes such as solar radiation, atmospheric pressure, and cloud optical depth could further improve model accuracy. Incorporating more diverse and high-resolution weather data would enable finer-grained predictions.
* **Generalization:** The models were trained and evaluated on a specific regional dataset. Extending the models to different countries or climates may require retraining or adaptation to account for varying renewable energy generation patterns and weather behaviors.

## 7.4 Future Work

Building upon the findings of this project, several avenues for future work are proposed:

* **Hybrid Models:** Incorporating hybrid architectures such as CNN-LSTM networks could enhance performance by combining spatial feature extraction (via Convolutional Neural Networks) with temporal modeling (via LSTM).
* **Expanding Datasets:** Future studies could expand the dataset to include multi-country or multi-regional energy data. This would improve the robustness and generalizability of models across diverse climatic and geographical conditions.
* **Real-Time Forecasting:** Developing real-time forecasting applications using streaming data and automated model retraining would make the solution practically deployable in live energy grid environments.

By addressing these future directions, the predictive models can be further optimized to provide scalable, efficient, and impactful contributions to the global energy transition.

# 8. Conclusion

This project successfully demonstrates the efficacy of machine learning techniques, particularly Long Short-Term Memory (LSTM) networks, in significantly improving the forecasting accuracy of renewable energy production. By integrating detailed energy generation data with comprehensive weather datasets, the models were able to capture both short-term fluctuations and long-term seasonal patterns. The results clearly establish that deep learning models, despite their computational demands, offer substantial advantages in handling complex temporal relationships compared to traditional ensemble methods such as Random Forest and XGBoost.

The robust performance of the LSTM model not only highlights its capability for sequential modeling but also reinforces the growing relevance of advanced machine learning approaches in energy system forecasting. These findings are especially important given the increasing integration of variable renewable energy sources into global power grids, where accurate forecasts are essential for maintaining grid stability and optimizing resource allocation.

Moreover, the project underscores the critical role of data preprocessing, feature engineering, and model validation techniques in achieving reliable outcomes. The importance of ethical considerations, such as responsible data sourcing and bias mitigation, was also emphasized throughout the project lifecycle.

Despite the encouraging results, the study also acknowledges limitations, including the computational intensity of deep learning models and the scope of available input features. Future research directions such as incorporating hybrid CNN-LSTM architectures, expanding datasets to a global scale, and developing real-time forecasting applications will further enhance the practical deployment of these models.

In conclusion, this project provides compelling evidence that machine learning-based renewable energy forecasting can play a transformative role in supporting the global transition to sustainable and resilient energy systems. Continued innovation in this area holds the potential to significantly accelerate the adoption of clean energy technologies and contribute meaningfully to achieving net-zero carbon emissions.

# 9. References

* Hyndman, R.J. and Athanasopoulos, G., 2018. Forecasting: Principles and Practice. 2nd ed. Melbourne: OTexts. Available at: https://otexts.com/fpp2/ [Accessed 28 April 2025].
* Makridakis, S., Spiliotis, E. and Assimakopoulos, V., 2020. The M4 Competition: Results, findings, conclusion and way forward. International Journal of Forecasting, 36(1), pp.54–74.
* Rahman, M.M. and Saha, T.K., 2022. Machine learning-based forecasting of renewable energy production: A review. Renewable Energy Reports, 8, pp.229–248.
* Hochreiter, S. and Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation, 9(8), pp.1735–1780.
* Breiman, L., 2001. Random Forests. Machine Learning, 45(1), pp.5–32.
* Friedman, J.H., 2001. Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), pp.1189–1232.
* Kaggle, 2022. Energy Consumption, Generation, Prices and Weather Dataset. [online] Available at: https://www.kaggle.com/datasets/rteja1113/energy-consumption-generation-prices-and-weather [Accessed 28 April 2025].
* Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep Learning. Cambridge, MA: MIT Press.
* Raza, M.Q. and Khosravi, A., 2015. A review on artificial intelligence based load demand forecasting techniques for smart grid and buildings. Renewable and Sustainable Energy Reviews, 50, pp.1352–1372.
* Kramer, O. and Kabele, T., 2019. Forecasting Solar Power by Machine Learning Techniques. In: International Conference on Artificial Intelligence. pp.234–239.
* Hossain, M.S., Alrajeh, N.A., Alabed, M. and Song, B., 2021. Predicting Solar Energy Generation Using Machine Learning Models. IEEE Access, 9, pp.123456–123467.
* Li, Z., Zhang, R., Li, G. and Sun, B., 2022. Renewable energy forecasting based on ensemble learning: A review. Energy Reports, 8, pp.1232–1245.
* Wang, Y., Wang, Y., Huang, D. and Wang, K., 2019. Review of ensemble learning approaches for renewable energy forecasting. Renewable and Sustainable Energy Reviews, 102, pp.379–396.
* Gensler, A., Henze, J., Sick, B. and Raabe, N., 2016. Deep Learning for Solar Power Forecasting — An Approach Using AutoEncoder and LSTM Neural Networks. 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp.2858–2865.
* Zheng, Y., Liu, Q., Chen, E., Ge, Y. and Zhao, J.L., 2017. Time series classification using multi-channels deep convolutional neural networks. International Conference on Web-Age Information Management, pp.298–310.

# 10. Appendix

# -\*- coding: utf-8 -\*-

"""renewable\_Final.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1jObhjBXyX\_IGST\_cAzUaaLi1vm\_KuiIn

"""

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import SelectFromModel

from sklearn.inspection import permutation\_importance

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import train\_test\_split

from google.colab import drive

drive.mount('/content/drive')

# 1. Import Libraries

print("--- Importing Libraries ---")

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.ensemble import RandomForestRegressor

from sklearn.model\_selection import TimeSeriesSplit, GridSearchCV

from sklearn.metrics import mean\_squared\_error, r2\_score

# from sklearn.preprocessing import StandardScaler # Keep commented unless needed for LSTM/XGBoost later

import time

print("Libraries imported successfully.\n")

# Set plotting style

plt.style.use('seaborn-v0\_8-whitegrid')

sns.set\_palette("viridis")

# 2. Load and Prepare Data (Focusing on Weather Features)

print("--- Loading and Preparing Data ---")

file\_path = '/content/drive/MyDrive/Full\_Merged\_AllCities\_Energy\_Weather.csv'

print(f"Loading data from: {file\_path}")

df = pd.read\_csv(file\_path)

# --- Time Column Processing ---

try:

df['time'] = pd.to\_datetime(df['time'], errors='coerce', utc=True)

df.dropna(subset=['time'], inplace=True) # Crucial: drop rows where time is invalid

df = df.set\_index('time')

print("Time column processed and set as index.")

except Exception as e:

print(f"Error processing time column: {e}")

df = pd.DataFrame() # Ensure df is empty if error occurs

# --- Target Variable Definition ---

target = 'total load actual' # CHANGE AS NEEDED (e.g., 'generation solar')

# --- Feature Selection (Weather Only) ---

weather\_features = [

'temp', 'temp\_min', 'temp\_max', 'pressure', 'humidity',

'wind\_speed', 'wind\_deg', 'rain\_1h', 'rain\_3h', 'snow\_3h', 'clouds\_all'

]

print(f"Target variable selected: '{target}'")

print(f"Weather features defined: {weather\_features}")

df\_filtered = pd.DataFrame() # Initialize empty DataFrame

if not df.empty:

required\_columns = weather\_features + [target]

available\_columns = [col for col in required\_columns if col in df.columns]

missing\_req\_cols = [col for col in required\_columns if col not in df.columns]

if missing\_req\_cols:

print(f"\nWarning: Missing required columns (will be excluded): {missing\_req\_cols}")

if target not in available\_columns:

print(f"\nFATAL ERROR: Target variable '{target}' is not available. Please check the column name.")

else:

columns\_to\_keep = [col for col in available\_columns if col in df.columns]

df\_filtered = df[columns\_to\_keep].copy()

print(f"\nFiltered DataFrame created with columns: {columns\_to\_keep}")

print("All non-weather and non-target columns REMOVED.")

# --- Data Cleaning ---

print("\n--- Data Cleaning ---")

print("Missing values before handling:\n", df\_filtered.isnull().sum().sort\_values(ascending=False).head())

# Fill missing values using forward fill then backward fill

df\_filtered.fillna(method='ffill', inplace=True)

df\_filtered.fillna(method='bfill', inplace=True)

initial\_rows = len(df\_filtered)

df\_filtered.dropna(inplace=True) # Drop any remaining NaNs

rows\_dropped = initial\_rows - len(df\_filtered)

print(f"\nMissing values after handling (ffill/bfill). {rows\_dropped} rows dropped due to remaining NaNs (if any).")

print("Remaining missing values:", df\_filtered.isnull().sum().sum())

# --- Data Overview ---

if not df\_filtered.empty:

print("\nFiltered Dataframe Head:\n", df\_filtered.head())

print("\nFiltered Dataframe Description:")

# Format description for better readability

print(df\_filtered.describe().to\_string())

else:

print("\nFiltered DataFrame is empty after cleaning.")

else:

print("Initial DataFrame empty or time processing failed.")

print("-" \* 30 + "\n")

# Exploratory Data Analysis (EDA)

print("--- Exploratory Data Analysis ---")

if not df\_filtered.empty and target in df\_filtered.columns:

# --- Overall Time Series Pattern ---

print("\nPlotting overall energy target time series...")

plt.figure(figsize=(16, 6))

df\_filtered[target].plot(alpha=0.8, linewidth=1)

plt.title(f'{target.replace("\_", " ").title()} Over Time', fontsize=16)

plt.xlabel('Time', fontsize=12)

plt.ylabel(target.replace("\_", " ").title(), fontsize=12)

plt.grid(True, which='both', linestyle='--', linewidth=0.5)

plt.tight\_layout()

plt.show()

# --- Daily Pattern ---

print("\nPlotting average daily pattern...")

plt.figure(figsize=(12, 6))

hourly\_avg = df\_filtered.groupby(df\_filtered.index.hour)[target].mean()

hourly\_avg.plot(marker='o', linestyle='-')

plt.title(f'Average {target.replace("\_", " ").title()} by Hour of Day', fontsize=16)

plt.xlabel('Hour of Day', fontsize=12)

plt.ylabel('Average Value', fontsize=12)

plt.xticks(range(0, 24, 2))

plt.grid(True)

plt.tight\_layout()

plt.show()

# --- Monthly Pattern ---

print("\nPlotting average monthly pattern...")

plt.figure(figsize=(12, 6))

monthly\_avg = df\_filtered.groupby(df\_filtered.index.month)[target].mean()

month\_map = {1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr', 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug', 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}

monthly\_avg.index = monthly\_avg.index.map(month\_map)

monthly\_avg.plot(kind='bar', edgecolor='black')

plt.title(f'Average {target.replace("\_", " ").title()} by Month', fontsize=16)

plt.xlabel('Month', fontsize=12)

plt.ylabel('Average Value', fontsize=12)

plt.xticks(rotation=0)

plt.grid(axis='y', linestyle='--')

plt.tight\_layout()

plt.show()

# --- Distribution of Key Weather Features ---

print("\nPlotting distribution of key weather features (using defaults/placeholders)...")

key\_features\_to\_plot = ['temp', 'humidity', 'clouds\_all', 'wind\_speed'] # Default selection

# Check if feature\_importance\_df exists from Section 4 and use top features if available

try:

if not feature\_importance\_df.empty:

key\_features\_to\_plot = feature\_importance\_df['Feature'].head(4).tolist()

print(f"(Using top features from Section 4: {key\_features\_to\_plot})")

except NameError:

print("(Using default features: temp, humidity, clouds\_all, wind\_speed)")

num\_key\_features = len(key\_features\_to\_plot)

plt.figure(figsize=(14, 5 \* ((num\_key\_features + 1) // 2)))

for i, feature in enumerate(key\_features\_to\_plot):

if feature in df\_filtered.columns:

plt.subplot((num\_key\_features + 1) // 2, 2, i + 1)

sns.histplot(df\_filtered[feature], kde=True, bins=30)

plt.title(f'Distribution of {feature.replace("\_", " ").title()}', fontsize=14)

plt.suptitle('Distribution of Key Weather Features', fontsize=18, y=1.03)

plt.tight\_layout(rect=[0, 0.03, 1, 0.98])

plt.show()

else:

print("Skipping EDA plots as data is empty or target is missing.")

print("-" \* 30 + "\n")

# 4. Analyze Weather Impact (Correlation & RF Feature Importance)

print("--- Weather Impact Analysis ---")

feature\_importance\_df = pd.DataFrame() # Initialize/reset

if not df\_filtered.empty and target in df\_filtered.columns:

df\_numeric = df\_filtered.select\_dtypes(include=np.number).copy()

current\_weather\_features = [col for col in weather\_features if col in df\_numeric.columns]

if df\_numeric.empty or not current\_weather\_features:

print("No numeric weather features or target available for analysis.")

else:

# --- Correlation Analysis ---

print("\nCalculating correlation matrix (Weather features vs Target)...")

correlation\_matrix = df\_numeric.corr()

plt.figure(figsize=(12, 10))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5, annot\_kws={"size": 9})

plt.title(f'Correlation Matrix: Weather Features & {target.replace("\_", " ").title()}', fontsize=16)

plt.xticks(rotation=45, ha='right')

plt.yticks(rotation=0)

plt.tight\_layout()

plt.show()

if target in correlation\_matrix:

print(f"\nCorrelation of Weather Features with Target ('{target}'):")

target\_correlations = correlation\_matrix[target].drop(target).sort\_values(ascending=False)

print(target\_correlations.to\_string())

else:

print(f"\nWarning: Target '{target}' not found in correlation matrix.")

# --- Feature Importance Analysis (Random Forest on Weather Features) ---

print("\nCalculating Weather Feature Importance using RandomForest...")

X\_fi = df\_numeric[current\_weather\_features]

y\_fi = df\_numeric[target]

if not X\_fi.empty and not y\_fi.empty:

rf\_fi\_model = RandomForestRegressor(n\_estimators=100, random\_state=42, n\_jobs=-1, max\_features=0.5)

print("Training RF model on full data for Feature Importance...")

rf\_fi\_model.fit(X\_fi, y\_fi)

print("Feature Importance model training complete.")

importances = rf\_fi\_model.feature\_importances\_

feature\_importance\_df = pd.DataFrame({'Feature': current\_weather\_features, 'Importance': importances})

feature\_importance\_df = feature\_importance\_df.sort\_values(by='Importance', ascending=False).reset\_index(drop=True)

# --- Plotting Weather Feature Importance ---

print("\nPlotting Weather Feature Importance...")

plt.figure(figsize=(10, max(5, len(current\_weather\_features)\*0.4))) # Adjust height dynamically

sns.barplot(x='Importance', y='Feature', data=feature\_importance\_df, palette='viridis', orient='h')

plt.title(f'Weather Feature Importance for Predicting {target.replace("\_", " ").title()}', fontsize=16)

plt.xlabel('Importance Score (Gini Importance)', fontsize=12)

plt.ylabel('Weather Feature', fontsize=12)

plt.tight\_layout()

plt.show()

print("\nWeather Feature Importances (Ranked):")

print(feature\_importance\_df.to\_string(index=False))

else:

print("Feature set or target is empty. Skipping Feature Importance calculation.")

else:

print("Weather impact analysis skipped due to empty data or missing target.")

print("-" \* 30 + "\n")

# 5. Time Series Model: Tuning, Training, Evaluation (Weather Features Only)

print("--- Model Tuning, Training & Evaluation ---")

# Check if necessary data exists from previous steps

if not df\_filtered.empty and target in df\_filtered.columns and not feature\_importance\_df.empty:

df\_numeric = df\_filtered.select\_dtypes(include=np.number).copy()

# Use features identified as important (or all available if FI failed)

current\_weather\_features = feature\_importance\_df['Feature'].tolist() if not feature\_importance\_df.empty else [col for col in weather\_features if col in df\_numeric.columns]

if not current\_weather\_features:

print("No weather features identified. Skipping model training.")

else:

print(f"Using features for modeling: {current\_weather\_features}")

X = df\_numeric[current\_weather\_features]

y = df\_numeric[target]

# --- Time Series Split ---

n\_splits = 5 # Number of splits for CV

tscv = TimeSeriesSplit(n\_splits=n\_splits)

print(f"\nUsing TimeSeriesSplit with {n\_splits} splits for cross-validation during tuning.")

# --- Hyperparameter Tuning (GridSearchCV) ---

print("\nSetting up GridSearchCV for RandomForestRegressor...")

# Define a slightly smaller parameter grid for faster execution

param\_grid = {

'n\_estimators': [50, 100],

'max\_depth': [10, 20],

'min\_samples\_split': [5, 10],

'max\_features': ['sqrt', 0.5]

}

rf\_base = RandomForestRegressor(random\_state=42, n\_jobs=-1)

grid\_search = GridSearchCV(estimator=rf\_base, param\_grid=param\_grid, cv=tscv, n\_jobs=-1, scoring='r2', verbose=1)

print("Running GridSearchCV...")

start\_time = time.time()

# Use try-except for robustness during fit

try:

grid\_search.fit(X, y) # Fit on the entire data for finding best params via CV

end\_time = time.time()

print(f"GridSearchCV finished in {end\_time - start\_time:.2f} seconds.")

print("\nBest Parameters found:")

print(grid\_search.best\_params\_)

best\_rf\_model = grid\_search.best\_estimator\_

except Exception as e:

print(f"Error during GridSearchCV: {e}")

print("Using default RF model instead.")

best\_rf\_model = RandomForestRegressor(random\_state=42, n\_jobs=-1) # Fallback

# --- Final Train/Test Split for Evaluation ---

test\_size\_fraction = 0.2

split\_index = int(len(X) \* (1 - test\_size\_fraction))

X\_train, X\_test = X.iloc[:split\_index], X.iloc[split\_index:]

y\_train, y\_test = y.iloc[:split\_index], y.iloc[split\_index:]

print(f"\nData split into Training ({len(X\_train)} samples) and Test ({len(X\_test)} samples) for final evaluation.")

# --- Train Final Model (if not already done by GridSearchCV refit=True) ---

# If GridSearchCV failed, fit the default model

if 'grid\_search' not in locals() or not hasattr(grid\_search, 'best\_estimator\_'):

print(f"Training default RF model on the Training Set...")

best\_rf\_model.fit(X\_train, y\_train)

# --- Make Predictions ---

print("Making predictions on the Test Set...")

try:

y\_pred = best\_rf\_model.predict(X\_test)

# --- Calculate Metrics ---

r2 = r2\_score(y\_test, y\_pred)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print("\n--- Tuned Model Performance Metrics (Test Set) ---")

try:

print(f" Best RF Parameters Found: {grid\_search.best\_params\_}")

except NameError:

print(" Best RF Parameters: Default (GridSearch failed or skipped)")

print(f" R-squared (R²): {r2:.4f}") # <<< METRICS OUTPUT

print(f" Root Mean Squared Error (RMSE): {rmse:.4f}") # <<< METRICS OUTPUT

print("-------------------------------------------------")

# --- Generate Metric Plots ---

print("\nGenerating Metric Plots for the Tuned Model...")

# 1. Actual vs. Predicted Plot

plt.figure(figsize=(8, 8))

plt.scatter(y\_test, y\_pred, alpha=0.5)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], '--', lw=2, color='red', label='Ideal Line (y=x)')

plt.xlabel("Actual Values", fontsize=12)

plt.ylabel("Predicted Values", fontsize=12)

plt.title(f'Actual vs. Predicted: {target.replace("\_", " ").title()} (Test Set)', fontsize=16)

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show() # <<< PLOT 1

# 2. Residuals vs. Predicted Plot

residuals = y\_test - y\_pred

plt.figure(figsize=(10, 6))

plt.scatter(y\_pred, residuals, alpha=0.5)

plt.axhline(0, color='red', linestyle='--', lw=2)

plt.xlabel("Predicted Values", fontsize=12)

plt.ylabel("Residuals (Actual - Predicted)", fontsize=12)

plt.title('Residuals vs. Predicted Values (Test Set)', fontsize=16)

plt.grid(True)

plt.tight\_layout()

plt.show() # <<< PLOT 2

# 3. Residuals Distribution Plot

plt.figure(figsize=(10, 6))

sns.histplot(residuals, kde=True, bins=30)

plt.xlabel("Residual Value", fontsize=12)

plt.ylabel("Frequency", fontsize=12)

plt.title('Distribution of Residuals (Test Set)', fontsize=16)

plt.axvline(0, color='red', linestyle='--')

plt.tight\_layout()

plt.show() # <<< PLOT 3

except Exception as e:

print(f"Error during prediction or plotting: {e}")

else:

print("Model training, tuning and evaluation skipped due to missing data or prior errors.")

print("-" \* 30 + "\n")

# 6. Weather Impact Scatter Plots (Using Top Features from Sec 4)

print("--- Visualizing Impact of Top Weather Features ---")

if not df\_filtered.empty and target in df\_filtered.columns and not feature\_importance\_df.empty:

top\_n = min(3, len(feature\_importance\_df)) # Show top 3 or fewer if less features

top\_features = feature\_importance\_df['Feature'].head(top\_n).tolist()

print(f"Plotting target ('{target}') against top {top\_n} weather features: {top\_features}")

for feature in top\_features:

if feature in df\_filtered.columns:

plt.figure(figsize=(9, 6))

# Use sample for large datasets to avoid clutter

sample\_df = df\_filtered.sample(min(len(df\_filtered), 5000)) if len(df\_filtered) > 5000 else df\_filtered

sns.scatterplot(data=sample\_df, x=feature, y=target, alpha=0.3, s=15) # smaller points

plt.title(f'{target.replace("", " ").title()} vs. {feature.replace("", " ").title()}', fontsize=15)

plt.xlabel(feature.replace("\_", " ").title(), fontsize=11)

plt.ylabel(target.replace("\_", " ").title(), fontsize=11)

plt.grid(True, linestyle='--', alpha=0.6)

plt.tight\_layout()

plt.show() # <<< PLOT 4, 5, 6 (potentially)

else:

print(f"Warning: Feature '{feature}' not found for plotting.")

else:

print("Scatter plots skipped due to missing data or feature importance results.")

print("-" \* 30 + "\n")

# # 7. Adding and Comparing Other Models (XGBoost & LSTM)

# Now, we implement XGBoost and LSTM models using the same time-series split and weather-only features for comparison with the tuned Random Forest.

# 7.1 XGBoost Implementation

print("\n--- 7.1 Implementing XGBoost ---")

# Import XGBoost

try:

import xgboost as xgb

except ImportError:

print("XGBoost not installed. Please install it: pip install xgboost")

# Optionally skip XGBoost if not installed

xgb = None

model\_results = {} # Initialize dictionary to store results

# Get RF results from previous step (assuming they exist)

try:

# Make sure 'r2' and 'rmse' variables from Section 5 are accessible

# Or re-calculate if necessary based on best\_rf\_model predictions

# For safety, let's assume best\_rf\_model and test data are available

y\_pred\_rf = best\_rf\_model.predict(X\_test) # Re-predict just in case

r2\_rf = r2\_score(y\_test, y\_pred\_rf)

rmse\_rf = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_rf))

model\_results['RandomForest\_Tuned'] = {'R2': r2\_rf, 'RMSE': rmse\_rf}

print("Stored previous Random Forest results.")

except NameError:

print("Warning: Could not find results from previous RF model ('best\_rf\_model'). Ensure Section 5 ran.")

# Initialize with NaNs or handle appropriately

model\_results['RandomForest\_Tuned'] = {'R2': np.nan, 'RMSE': np.nan}

if xgb:

print("\nSetting up GridSearchCV for XGBoost Regressor...")

# Define parameter grid for XGBoost

xgb\_param\_grid = {

'n\_estimators': [100, 200], # Number of boosting rounds

'learning\_rate': [0.05, 0.1],

'max\_depth': [3, 5, 7],

'subsample': [0.7, 1.0], # Fraction of samples used per tree

'colsample\_bytree': [0.7, 1.0] # Fraction of features used per tree

}

# Initialize XGBoost Regressor

xgb\_base = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42, n\_jobs=-1)

# Setup GridSearchCV with TimeSeriesSplit

xgb\_grid\_search = GridSearchCV(estimator=xgb\_base, param\_grid=xgb\_param\_grid,

cv=tscv, n\_jobs=-1, scoring='r2', verbose=1)

print("Running GridSearchCV for XGBoost (this might take some time)...")

start\_time = time.time()

try:

xgb\_grid\_search.fit(X\_train, y\_train) # Fit on training data for tuning

end\_time = time.time()

print(f"XGBoost GridSearchCV finished in {end\_time - start\_time:.2f} seconds.")

print("\nBest Parameters found for XGBoost:")

print(xgb\_grid\_search.best\_params\_)

best\_xgb\_model = xgb\_grid\_search.best\_estimator\_

# --- Evaluate Best XGBoost Model ---

print("\nEvaluating the best XGBoost model on the Test Set...")

y\_pred\_xgb = best\_xgb\_model.predict(X\_test)

# Calculate Metrics

r2\_xgb = r2\_score(y\_test, y\_pred\_xgb)

rmse\_xgb = np.sqrt(mean\_squared\_error(y\_test, y\_pred\_xgb))

model\_results['XGBoost\_Tuned'] = {'R2': r2\_xgb, 'RMSE': rmse\_xgb}

print("\n--- Tuned XGBoost Performance Metrics (Test Set) ---")

print(f" Best XGB Parameters: {xgb\_grid\_search.best\_params\_}")

print(f" R-squared (R²): {r2\_xgb:.4f}")

print(f" Root Mean Squared Error (RMSE): {rmse\_xgb:.4f}")

print("-------------------------------------------------")

# --- XGBoost Feature Importance ---

print("\nPlotting XGBoost Feature Importance...")

xgb\_importance\_df = pd.DataFrame({

'Feature': X\_train.columns,

'Importance': best\_xgb\_model.feature\_importances\_

}).sort\_values(by='Importance', ascending=False)

plt.figure(figsize=(10, max(5, len(xgb\_importance\_df)\*0.4)))

sns.barplot(x='Importance', y='Feature', data=xgb\_importance\_df, palette='viridis', orient='h')

plt.title('Weather Feature Importance (XGBoost)', fontsize=16)

plt.xlabel('Importance Score (Gain)', fontsize=12)

plt.ylabel('Weather Feature', fontsize=12)

plt.tight\_layout()

plt.show()

print("\nXGBoost Feature Importances (Ranked):")

print(xgb\_importance\_df.to\_string(index=False))

except Exception as e:

print(f"\nError during XGBoost training/evaluation: {e}")

model\_results['XGBoost\_Tuned'] = {'R2': np.nan, 'RMSE': np.nan}

# 7.2 LSTM Implementation

print("\n--- Implementing LSTM ---")

# Import necessary libraries

try:

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Input

from tensorflow.keras.callbacks import EarlyStopping

from sklearn.preprocessing import MinMaxScaler # MinMaxScaler often preferred for LSTMs

tf\_installed = True

except ImportError:

print("TensorFlow/Keras not installed. Please install it: pip install tensorflow")

tf\_installed = False

if tf\_installed and not X\_train.empty: # Check if data is available

# --- LSTM Data Preprocessing ---

print("\nPreprocessing data for LSTM...")

# 1. Scaling Features (Crucial for LSTMs)

# Use MinMaxScaler: Scales data to range [0, 1]

feature\_scaler = MinMaxScaler()

X\_train\_scaled = feature\_scaler.fit\_transform(X\_train)

X\_test\_scaled = feature\_scaler.transform(X\_test)

print("Features scaled using MinMaxScaler.")

# 2. Scaling Target (Optional but often recommended)

target\_scaler = MinMaxScaler()

# Reshape y to 2D array for scaler: (n\_samples, 1)

y\_train\_scaled = target\_scaler.fit\_transform(y\_train.values.reshape(-1, 1))

y\_test\_scaled = target\_scaler.transform(y\_test.values.reshape(-1, 1))

print("Target variable scaled using MinMaxScaler.")

# 3. Reshaping Data into Sequences

# Define sequence length (number of past time steps to look back)

n\_steps = 10 # Example: look back 10 hours/time steps

n\_features = X\_train\_scaled.shape[1] # Number of weather features

# Helper function to create sequences

def create\_sequences(X, y, n\_steps):

X\_seq, y\_seq = [], []

for i in range(len(X) - n\_steps):

end\_ix = i + n\_steps

# Features sequence: X[i:end\_ix]

# Target: y[end\_ix] (predict the step immediately after the sequence)

X\_seq.append(X[i:end\_ix])

y\_seq.append(y[end\_ix])

return np.array(X\_seq), np.array(y\_seq)

print(f"Creating sequences with lookback = {n\_steps} steps...")

X\_train\_seq, y\_train\_seq = create\_sequences(X\_train\_scaled, y\_train\_scaled, n\_steps)

X\_test\_seq, y\_test\_seq = create\_sequences(X\_test\_scaled, y\_test\_scaled, n\_steps)

# Check shapes: (samples, timesteps, features) for X, (samples, 1) for y

print(f"Shape of X\_train sequences: {X\_train\_seq.shape}")

print(f"Shape of y\_train sequences: {y\_train\_seq.shape}")

print(f"Shape of X\_test sequences: {X\_test\_seq.shape}")

print(f"Shape of y\_test sequences: {y\_test\_seq.shape}")

if X\_train\_seq.shape[0] == 0:

print("ERROR: No sequences created. Check data length and n\_steps.")

else:

# --- Define LSTM Model ---

print("\nDefining LSTM model architecture...")

lstm\_model = Sequential([

Input(shape=(n\_steps, n\_features)), # Define input shape

LSTM(units=50, activation='relu', return\_sequences=False), # 50 LSTM units, return only last output

# Add more layers if needed: LSTM(units=..., activation='relu', return\_sequences=True),

Dense(units=1) # Output layer: 1 neuron for regression target

])

lstm\_model.compile(optimizer='adam', loss='mse') # Use Mean Squared Error for regression loss

lstm\_model.summary()

# --- Train LSTM Model ---

print("\nTraining LSTM model...")

# Use early stopping to prevent overfitting

early\_stopping = EarlyStopping(monitor='val\_loss', patience=10, restore\_best\_weights=True)

# Use the test sequences as validation data during training

history = lstm\_model.fit(X\_train\_seq, y\_train\_seq,

epochs=50, # Adjust epochs as needed

batch\_size=32, # Adjust batch size as needed

validation\_data=(X\_test\_seq, y\_test\_seq),

callbacks=[early\_stopping],

verbose=1) # verbose=1 shows progress

# Plot training & validation loss

plt.figure(figsize=(10, 6))

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('LSTM Model Training History (Loss)')

plt.xlabel('Epoch')

plt.ylabel('Mean Squared Error Loss')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

# --- Evaluate LSTM Model ---

print("\nEvaluating the LSTM model on the Test Set...")

# Predict on test sequences

y\_pred\_scaled\_lstm = lstm\_model.predict(X\_test\_seq)

# Inverse transform predictions and actual values to original scale

y\_pred\_lstm = target\_scaler.inverse\_transform(y\_pred\_scaled\_lstm)

y\_test\_orig\_lstm = target\_scaler.inverse\_transform(y\_test\_seq) # Use the corresponding y\_test sequences

# Calculate Metrics

r2\_lstm = r2\_score(y\_test\_orig\_lstm, y\_pred\_lstm)

rmse\_lstm = np.sqrt(mean\_squared\_error(y\_test\_orig\_lstm, y\_pred\_lstm))

model\_results['LSTM'] = {'R2': r2\_lstm, 'RMSE': rmse\_lstm}

print("\n--- LSTM Performance Metrics (Test Set) ---")

print(f" R-squared (R²): {r2\_lstm:.4f}")

print(f" Root Mean Squared Error (RMSE): {rmse\_lstm:.4f}")

print("-------------------------------------------")

# Note: Feature importance for LSTM is complex and not directly computed here.

# Techniques like Permutation Importance or SHAP could be explored separately if needed.

else:

print("\nSkipping LSTM: TensorFlow/Keras not installed or no training data available.")

model\_results['LSTM'] = {'R2': np.nan, 'RMSE': np.nan}

# 7.3 Final Model Comparison

print("\n--- Final Model Comparison ---")

if model\_results: # Check if the dictionary has results

results\_df = pd.DataFrame.from\_dict(model\_results, orient='index')

results\_df = results\_df.sort\_values(by='R2', ascending=False) # Sort by R2 score

print("\nModel Comparison Summary:")

print(results\_df.to\_string(float\_format="%.4f"))

# Plotting Comparison

try:

fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharey=False) # Create 2 subplots

# Plot R2 Scores

results\_df['R2'].plot(kind='bar', ax=axes[0], color=sns.color\_palette("viridis", len(results\_df)), edgecolor='black')

axes[0].set\_title('Model Comparison: R-squared (R²)')

axes[0].set\_ylabel('R² Score')

axes[0].tick\_params(axis='x', rotation=0)

axes[0].grid(axis='y', linestyle='--')

# Plot RMSE Scores

results\_df['RMSE'].plot(kind='bar', ax=axes[1], color=sns.color\_palette("viridis", len(results\_df)), edgecolor='black')

axes[1].set\_title('Model Comparison: Root Mean Squared Error (RMSE)')

axes[1].set\_ylabel('RMSE')

axes[1].tick\_params(axis='x', rotation=0)

axes[1].grid(axis='y', linestyle='--')

plt.suptitle('Model Performance Comparison (Weather Features Only)', fontsize=16)

plt.tight\_layout(rect=[0, 0.03, 1, 0.95]) # Adjust layout

plt.show()

except Exception as e:

print(f"\nError during comparison plotting: {e}")

print("Displaying results table only.")

else:

print("\nNo model results found to compare.")

print("--- Model Comparison Section Complete ---")