Goldsmiths
UNIVERSITY OF LONDON

Developing a Machine Learning Model for Short-Term Solar Energy Forecasting using Spark/MLlib

A Case Study with Enefit in Estonia

A Thesis Submitted In Fulfilment of the Requirements for the Degree of

MASTERS In Data Science and Artificial Intelligence

To the

Department of Computing

GOLDSMITHS, UNIVERSITY OF LONDON

New Cross, London SE14 6NW

September 2024

Under the Supervision of Dr V L Raju Chinthalapati

Carlos Manuel De Oliveira Alves

Student ID: 33617310

CERTIFICATE

September 2024

This is to certify that the work embodied in the thesis entitled ***"Developing a Machine Learning Model for Short-Term Solar Energy Forecasting using Spark/MLlib A Case Study with Enefit in Estonia"*** done by Carlos Manuel De Oliveira Alves, Student 33617310, as a Postgraduate student in the Department of Computing, Goldsmiths University of London, UK, is authentic. He carried out this work under my guidance.

This work is based on original research, and to the best of my knowledge and belief, the matter embodied in this research plan has not been submitted earlier for the award of any degree or diploma.

Supervisor

Dr V L Raju Chinthalapati
Senior Lecturer and Module Leader,
Department. of Computing,
Goldsmiths, University of London
London, United Kingdom
v.chinthalapati@gold.ac.uk

DECLARATION

I, Carlos Manuel De Oliveira Alves, Post-graduate student (Student ID: 33617310) in the Department of Computing, at this moment, declare that the synopsis titled ***"Developing a Machine Learning Model for Short-Term Solar Energy Forecasting using Spark/MLlib A Case Study with Enefit in Estonia"*** which is being submitted towards the fulfilment of the requirements for the degree of Masters in Data Science and Artificial Intelligence of Goldsmiths, University of London, United Kingdom is a record of bonafide research work carried out by me. I further declare that this work is based on original research and has not been submitted to any university or institution for any degree or diploma.

Carlos Manuel De Oliveira Alves
Student ID: 33617310
Dept. of Computing
Goldsmiths, University of London, United Kingdom
E-mail: cdeol003@gold.ac.uk

ACKNOWLEDGEMENT

After a hiatus of nearly three decades from formal education, I embarked on a transformative academic journey that has spanned five years. I first completed my Bachelor's in Computer Science and now have a Master's in Data Science and Artificial Intelligence. This experience has been nothing short of extraordinary, and I find myself marvelling at the timing uncertainty. As I conclude my studies, I feel I have arrived at precisely the right moment in the rapidly evolving field of AI.

As I reach the culmination of my Master's degree, I am filled with profound gratitude for the incredible journey that has brought me to this point. This achievement would not have been possible without numerous individuals and institutions' support, guidance, and encouragement.

To my fellow students and research colleagues: Your unwavering support and intellectual curiosity have been invaluable throughout this journey. Our stimulating discussions and late-night coding sessions enhanced our projects and fostered deep friendships. The collaborative spirit within our cohort has been a wellspring of motivation and continuous learning. I am incredibly grateful to Sandor Kanda, whose insights and encouragement have shaped my work and perspective.

To my family and friends: Your unconditional support and understanding during this challenging yet rewarding period have been my anchor. Your belief in me, especially as I returned to studies after such a long break, has been a constant source of strength.

As I look to the future, I am excited to apply the knowledge and skills I've gained and contribute to advancing AI and data science. This journey has reaffirmed that it's never too late to pursue one's passions and that remarkable transformations are possible with dedication and support.

Carlos Manuel De Oliveira Alves
September 2024

ABSTRACT

This thesis explores the application of machine learning techniques for short-term solar energy forecasting, addressing the critical challenge of integrating intermittent renewable energy sources into power grids. Utilising Apache Spark's MLlib library, the research implements and compares various algorithms, including linear regression, logistic regression, LSTM networks, CNNs, and MLPs, to predict solar energy output. The study aims to leverage distributed computing to process large-scale weather datasets efficiently, conducting feature engineering to identify key meteorological variables influencing solar energy production. Performance evaluation focuses on metrics such as RMSE and AUC-ROC, emphasising model optimisation through hyperparameter tuning and scalability assessment. By enhancing the accuracy of solar energy forecasting, this research contributes to improving renewable energy management, reducing energy imbalances, and supporting the transition to a sustainable energy future. The findings have potential implications for grid reliability, economic efficiency, and environmental sustainability in the rapidly evolving field of renewable energy.

TABLE OF CONTENTS

LIST OF TABLES, FIGURES, GRAPHS AND SCRIPTS

Tables:

Figures:

Scripts:

Graphs/Plots:

Scripts (continued):

Keywords — Solar Energy Forecasting, Machine Learning, Big Data Analytics, Renewable Energy, Time Series Prediction, Distributed Computing, Neural Networks, Hyperparameter Tuning, Feature Engineering, Weather Data Analysis, Energy Management Systems, Predictive Modelling, Model Evaluation Metrics, Ethical AI, Environmental Sustainability, Grid Stability, Climate Change Mitigation, Smart Grids, Data-Driven Decision Making, Geospatial Data Analysis

Chapter 1 - Introduction

1.1 Section Overview

Chapter 1 - Introduction
This chapter introduces the thesis topic, outlines the research's aims and objectives, and provides an overview of the document structure. It sets the context for the short-term solar energy forecasting study using machine learning techniques and Apache Spark's MLlib.

Chapter 2 - Background
This chapter presents a comprehensive literature review covering the current state of solar energy forecasting, machine learning applications in renewable energy, advanced techniques and hybrid models, and big data and distributed computing in energy forecasting. It also identifies gaps in the existing literature and suggests future research directions.

Chapter 3 - Ethical Considerations
This chapter discusses the ethical implications of using big data and machine learning in energy forecasting. It covers topics such as data privacy and security, algorithmic bias and fairness, environmental impact, transparency and explainability of models, and the technology's socioeconomic implications.

Chapter 4 - Methodology
This extensive chapter details the research methodology, including data sources, exploratory data analysis, and the implementation of various machine learning models using Spark/MLlib. It covers linear regression, logistic regression, LSTM networks, CNNs, and MLPs, explaining how each model is applied to the solar energy forecasting problem.

Chapter 5 – Discussion and Comparison of Results
This chapter compares the results obtained from different machine learning models and discusses each approach's performance, strengths, and limitations in the context of solar energy forecasting.

Chapter 6 – Conclusion
The final chapter summarizes the research's key findings, acknowledges the study's limitations, and proposes directions for future work in solar energy forecasting using machine learning techniques.

The appendices provide additional resources, including code repositories, academic references, and details on the programming languages, libraries, and frameworks used in the research.

1.2 Aims and Objectives

The global shift towards renewable energy sources has placed solar power at the forefront of sustainable energy solutions. As the world grapples with the urgent need to reduce carbon emissions and mitigate climate change, the role of solar energy becomes even more crucial. Solar energy presents a clean, abundant, and increasingly cost-effective alternative to fossil fuels. However, the intermittent nature of solar power generation poses significant challenges for grid integration and energy management. Accurate short-term forecasting of solar energy production has thus emerged as a critical factor in optimizing solar energy utilization and ensuring grid stability.

Recent advancements in machine learning and big data technologies have opened new avenues for improving the accuracy and efficiency of solar energy forecasting. By leveraging vast amounts of weather data and sophisticated algorithms, it is now possible to predict solar energy output with unprecedented precision. This capability not only enhances grid reliability but also contributes to reducing energy imbalances, potentially leading to significant economic and environmental benefits, underscoring the importance and impact of the research.

This thesis focuses on developing and evaluating machine learning models for short-term solar energy forecasting using Apache Spark's MLlib library. By harnessing the power of distributed computing, we aim to process and analyse large-scale weather datasets efficiently, uncovering valuable insights into the relationships between various meteorological variables and solar energy production.

The primary objectives of this research

1. To implement and compare different machine learning algorithms, including linear regression, logistic regression, Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNN), and Multi-Layer Perceptron's (MLP), for predicting short-term solar energy output.

2. To evaluate the performance of these models using metrics such as Root Mean Square Error (RMSE) and Area Under the Receiver Operating Characteristic Curve (AUC-ROC).

3. To conduct feature engineering to identify the most significant weather variables influencing solar energy production.

4. Optimize model performance through hyperparameter tuning and assess the scalability of using Apache Spark to process extensive weather datasets.

By addressing these objectives, this research aims to contribute to the growing knowledge of solar energy forecasting and provide practical insights for improving renewable energy management. The findings of this study have the potential to inform more efficient solar energy

integration strategies, reduce energy imbalances, and ultimately support the transition to a more sustainable energy future, instilling a sense of hope and optimism in the audience.

The subsequent chapters will detail the methodology employed, present the results of our analysis, discuss the implications of our findings, and propose directions for future research in this rapidly evolving field.

Chapter 2 - Background

2.1 Literature Review

Solar Energy Forecasting and Machine Learning Applications in Renewable Energy

The Current State of Solar Energy Forecasting

Solar energy forecasting is crucial in managing and integrating photovoltaic (PV) systems into the power grid. Accurate and reliable forecasting models are essential to handle solar energy's variability and intermittency, improving stability and operational efficiency.

Recent studies have demonstrated the effectiveness of hybrid and advanced machine learning models in improving solar energy forecasting:

Table 2.1: Solar Energy Forecasting Studies

Study	Authors (Year)	Model(s) Used	Key Findings
1	Nguyen et al. (2023)	Combined NARX-LSTM	Hybrid model outperformed standalone NARX and LSTM models in terms of mean absolute and square errors
2	Andrade et al. (2023)	MLP, RNN, LSTM	LSTM generally preferred, but MLPs offer computationally efficient alternative with comparable accuracy in specific scenarios
3	Mellit et al. (2021)	Deep Learning Neural Networks (DLNNs), including LSTM	DLNNs provide accurate short-term PV power forecasting, especially for one-step-ahead predictions
4	Chang et al. (2019)	WT-Adam-LSTM hybrid	Significantly improved electricity price prediction accuracy

1. Nguyen et al. (2023) investigated using combined NARX-LSTM models for solar energy forecasting [1]. They found that hybrid models can better capture the nonlinear patterns and dependencies in solar energy data, resulting in improved forecasting accuracy compared to standalone models. Their study, which collected data from a solar power plant over a year, demonstrated that the combined model outperformed individual NARX and LSTM models regarding mean absolute and square errors.

2. Andrade et al. (2023) compared the performance of MLP, RNN, and LSTM models for very short-term PV energy forecasting [2]. While LSTM models are generally preferred for their sequential data handling capabilities, MLPs offer a computationally efficient alternative with comparable accuracy in specific scenarios.

3. Mellit et al. (2021) developed various deep learning neural networks (DLNNs), including LSTM, for short-term forecasting of photovoltaic power [9]. Their study showed that DLNNs provide accurate short-term photovoltaic power forecasting, particularly for one-step-ahead predictions, with acceptable results for multi-step-ahead predictions.

4. Chang et al. (2019) introduced a WT-Adam-LSTM hybrid model, which combines a wavelet transform with an Adam-optimized LSTM neural network [3]. This model significantly improved electricity price prediction accuracy, which is closely related to energy production forecasting.

These studies highlight the trend towards hybrid and advanced neural network models in solar energy forecasting, demonstrating improvements over traditional methods.

Machine Learning Applications in Renewable Energy

Machine learning, profound learning models, has significantly advanced the field of renewable energy forecasting, with applications ranging from energy consumption to price prediction.

Table 2.2: Machine Learning in Renewable Energy Forecasting Studies

Authors (Year)	Model(s) Used	Application Area	Key Findings
Wibawa et al. (2023)	CNN, RNN, LSTM, Bi-LSTM	Energy usage forecasting	LSTM models outperformed others in capturing complex temporal dependencies, with better MAPE and RMSE.
AL-Ghamdi et al. (2023)	Hybrid DNN and multilayered LSTM	Household energy consumption prediction	Hybrid model outperformed standalone DNN and LSTM models, achieving R^2 of 0.99911.
Yan et al. (2019)	Hybrid LSTM with stationary wavelet transform	Individual household energy consumption forecasting	Superior performance compared to state-of-the-art methods.
Laib et al. (2019)	Hybrid LSTM	Natural gas consumption prediction	Significant improvements in forecasting accuracy compared to other techniques.
Rafik et al. (2020)	LSTM networks	Energy consumption prediction	Effectively predicted energy consumption, outperforming moving averages in various applications.
Zhou et al. (2020)	LSTM	Air-conditioning energy consumption prediction	Significant accuracy improvements over traditional models.
Succetti et al. (2020)	LSTM networks	Multivariate forecasting of energy time series	Effective in predicting energy time series by combining related series and considering long-term dependencies.
Wang et al. (2020)	LSTM	Periodic energy consumption prediction	Higher prediction performance than traditional methods for periodic energy consumption.

1. Wibawa et al. (2023) explored deep learning architectures, such as CNN, RNN, LSTM, and Bi-LSTM, to forecast energy usage [20]. They optimized the alpha value in exponential smoothing to enhance the accuracy of these models. The study demonstrated the effectiveness of LSTM models in capturing complex temporal dependencies, thus outperforming other

models in terms of accuracy metrics like mean absolute percentage error (MAPE) and root mean square error (RMSE).

2. AL-Ghamdi et al. (2023) presented a hybrid deep neural network (DNN) and multilayered LSTM model for household energy consumption prediction [16]. Their model outperformed standalone DNN and LSTM models, achieving a high coefficient of determination (R^2) of 0.99911. This indicates the potential of combining different deep-learning techniques to enhance prediction accuracy.

3. Yan et al. (2019) proposed a hybrid LSTM neural network combined with the stationary wavelet transform technique to improve forecasting accuracy for individual household energy consumption [1]. This approach demonstrated superior performance compared to state-of-the-art methods.

4. Laib et al. (2019) introduced a novel hybrid forecasting approach using LSTM for predicting natural gas consumption [2]. Their method showed significant improvements in forecasting accuracy compared to other techniques.

5. Rafik et al. (2020) discussed a new model for learning and predicting energy consumption using LSTM networks [4]. Their LSTM-based model effectively predicted energy consumption in various contexts, outperforming moving averages in various applications.

6. Zhou et al. (2020) demonstrated the effectiveness of LSTM in predicting the energy consumption of air-conditioning systems with significant accuracy improvements over traditional models [5].

7. Succetti et al. (2020) focused on using LSTM networks for multivariate forecasting of energy time series [6]. Their research showed that LSTM networks effectively predict energy time series by combining related time series and considering long-term dependencies.

8. Wang et al. (2020) proposed a novel approach using LSTM to predict periodic energy consumption [8]. Their LSTM network-based approach effectively predicted periodic energy consumption with higher prediction performance than traditional methods.

These studies collectively demonstrate the versatility and effectiveness of machine learning, particularly LSTM-based models, in various aspects of renewable energy forecasting and management.

Advanced Techniques and Hybrid Models

Recent research has explored more sophisticated approaches, combining different machine learning techniques or integrating them with other methods:

Table 2.3: Advanced Machine Learning Techniques in Energy Forecasting Studies

Authors (Year)	Advanced Technique(s)	Application Area	Key Findings
Leeraksakiat and Pora (2020)	LSTM with transfer learning	Energy consumption forecasting based on human occupancy	Transfer learning on LSTM improved occupancy forecasting accuracy by tracking behavior changes and adapting to new users.
Ciechulski & Osowski (2021)	Ensemble of LSTM models	High-precision, short-time load forecasting	Ensemble of five independent LSTM predictions reduced MAPE and RMSE errors by more than 6% in power system load forecasting.
Mahjoub et al. (2022)	Comparison of LSTM, GRU, and Drop-GRU	Short-term power consumption forecasting	LSTM provided better short-term power consumption forecasting with fewer prediction errors and finer precision than GRU and Drop-GRU methods.
Abraham et al. (2022)	Combination of LSTM and CNN	Energy consumption forecasting	Enhanced accuracy demonstrated using a fourteen-year hourly dataset from a Kaggle competition.
Gupta et al. (2023)	Integrated CNN and LSTM networks	Power grid usage forecasting	Achieved lower root mean square error for household energy consumption forecasting.

1. Leeraksakiat and Pora (2020) examined the use of LSTM with transfer learning to enhance the performance of energy consumption forecasting based on human occupancy [7]. Their research showed that transfer learning on LSTM networks improves occupancy forecasting accuracy by tracking behaviour changes and adapting to new users.

2. Ciechulski & Osowski (2021) focused on LSTM for high-precision, short-time load forecasting, demonstrating robustness and accuracy [10]. They found that an ensemble of five independent LSTM predictions reduced MAPE and RMSE errors by more than 6% in power system load forecasting.

3. Mahjoub et al. (2022) compared LSTM, GRU, and Drop-GRU models for short-term power consumption forecasting, emphasizing the superior accuracy of LSTM [12]. Their study showed LSTM provided better short-term power consumption forecasting with fewer prediction errors and finer precision than the GRU and Drop-GRU methods.

4. Abraham et al. (2022) explored a combination of LSTM and CNN for energy consumption forecasting, demonstrating enhanced accuracy using a fourteen-year hourly dataset from a Kaggle competition [13].

5. Gupta et al. (2023) integrated CNN and LSTM networks to forecast power grid usage effectively, achieving lower root mean square error for household energy consumption forecasting [17].

These studies highlight the trend towards more complex, hybrid models that leverage the strengths of different machine learning techniques to improve forecasting accuracy and reliability.

Big Data and Distributed Computing in Energy Forecasting

The integration of big data technologies and distributed computing frameworks has become increasingly important in energy forecasting, enabling the processing of large-scale datasets and real-time analysis:

Table 2.4: Big Data and Distributed Computing in Energy Forecasting

Technology/Study	Approach	Application Area	Key Benefits/Findings
Apache Spark and MLlib	Integration of distributed computing framework with machine learning library	Large-scale energy forecasting	- Efficient processing of large datasets - Real-time data analysis - Scalable and high-performance forecasting models - Enhanced computational efficiency and prediction accuracy
Liang & Saha (2022)	LSTM with UK's public intelligent meter data	Energy consumption prediction	- High forecasting accuracy under varying weather conditions - Demonstrated potential of using large-scale, real-world data
Shen et al. (2022)	Federated learning with LSTM models	Electricity consumption forecasting	- Improved generalization of forecasts - Aligns with carbon-neutral goals - Demonstrated potential of distributed learning techniques

1. Use of Apache Spark and MLlib: Integrating Apache Spark and its machine learning library, MLlib, in energy forecasting has enabled efficient processing of large datasets and real-time data analysis. This is crucial for deploying scalable and high-performance forecasting models. Apache Spark's ability to handle vast amounts of data in parallel allows for the training and application of complex models like LSTMs and CNNs on extensive energy datasets, enhancing both computational efficiency and prediction accuracy.

2. Liang & Saha (2022) used LSTM with the UK's public intelligent meter data, achieving high forecasting accuracy under varying weather conditions [14]. Their approach demonstrated the potential of using large-scale, real-world data for energy consumption prediction.

3. Shen et al. (2022) incorporated federated learning with LSTM models to improve the generalization of electricity consumption forecasts [15]. This approach aligns with carbon-neutral goals and demonstrates the potential of distributed learning techniques in energy forecasting.

These developments highlight the importance of big data technologies and distributed computing in enhancing energy forecasting models' scalability and real-time capabilities.

Gaps in the Literature and Future Research Directions

Table 2.5: Gaps in Energy Forecasting Literature and Future Research Directions

Research Gap	Current Status	Future Research Directions
Computational Efficiency and Scalability	- Focus on prediction accuracy - Resource-intensive models - Limited practical deployment in large-scale, real-time applications	- Develop more efficient algorithms - Optimize models for real-time processing - Explore hardware acceleration techniques
Data Integration	- Challenges in integrating diverse data sources - Hybrid models (e.g., NARX-LSTM, DNN-LSTM) show potential - Need for further optimization	- Develop advanced data fusion techniques - Optimize models for different types of energy data - Improve handling of varying temporal scales
Model Generalization	- Limited ability to generalize across different contexts - Models often specific to particular locations or consumption patterns	- Develop more robust, adaptable models - Investigate transfer learning techniques - Create standardized benchmarks across diverse scenarios
Long-term Forecasting	- Significant improvements in short-term forecasting - Long-term forecasting remains challenging	- Develop specialized models for long-term predictions - Investigate hybrid approaches combining short and long-term models - Incorporate more diverse, long-term influencing factors
Interpretability	- Increasing complexity of models - Limited explainability, especially for critical decisions	- Develop interpretable AI techniques for energy forecasting - Create visualization tools for model decisions - Investigate hybrid models combining interpretable and black-box components

Despite the advancements in forecasting models, several gaps still need to be addressed:

1. **Computational Efficiency and Scalability:** Current research often emphasizes prediction accuracy without addressing the resource-intensive nature of these models, which limits their practical deployment in large-scale, real-time energy data applications.

2. **Data Integration:** Integrating diverse data sources such as weather conditions, historical energy consumption, and real-time grid data remains challenging. While hybrid models like NARX-LSTM [1] and DNN-LSTM [16] have shown potential, further optimization is needed for different types of energy data and varying temporal scales.

3. **Model Generalization:** Further research is required to improve the ability of models to generalize across different geographic locations, climates, and energy consumption patterns.

4. **Long-term Forecasting:** While short-term forecasting has seen significant improvements, long-term forecasting of renewable energy production and consumption remains challenging and requires further investigation.

5. **Interpretability:** As models become more complex, there's a growing need for interpretable AI techniques to explain predictions, especially for critical energy management decisions.

Future research should focus on addressing these gaps by:
- Developing optimized machine learning models that improve prediction accuracy and computational efficiency.
- Leveraging big data processing frameworks like Apache Spark to handle large datasets and enable real-time data processing.
- Exploring novel hybrid models that can integrate diverse data sources and capture complex temporal and spatial dependencies.
- Investigating transfer learning and federated learning techniques to improve model generalization across different scenarios.
- Developing interpretable AI models for energy forecasting to enhance trust and adoption in critical energy management systems.

Conclusion
The reviewed literature highlights the critical role of accurate and efficient solar energy forecasting in integrating renewable energy sources into power grids. Significant progress has been made in developing advanced machine learning models, particularly in LSTM-based architectures and hybrid models. These advancements have improved forecasting accuracy for various aspects of renewable energy, including solar power production, energy consumption, and price prediction.

However, challenges persist regarding computational efficiency, data integration, and model generalization. Integrating big data technologies and distributed computing frameworks offers promising solutions to some of these challenges, enabling the processing of large-scale datasets and real-time analysis.

Addressing these gaps through optimized machine learning models, leveraging big data processing frameworks, and exploring novel hybrid approaches can significantly enhance the predictability and sustainability of renewable energy systems. As the field continues to evolve, the focus should be on developing models that are not only accurate but also computationally efficient, scalable, and interpretable, ensuring their effective deployment in real-world energy management scenarios.

Chapter 3 - Ethical Considerations

3.1 Introduction to Ethical Considerations
As we develop and implement advanced machine learning techniques for solar energy forecasting, we must consider the ethical implications of our research and its potential applications. This chapter explores the ethical considerations surrounding using big data, machine learning, and artificial intelligence in energy forecasting and management.

3.2 Data Privacy and Security

3.2.1 Data Collection and Storage

One of the primary ethical concerns in our research is collecting and storing large-scale weather and energy consumption data. While our dataset is primarily composed of meteorological information, it is essential to consider the following:

- There is potential for indirectly identifying individuals or communities through granular energy consumption patterns.
- The secure storage and data transmission to prevent unauthorized access or breaches.
- The retention period for collected data and protocols for data deletion when no longer needed.

Recommendation: Implement robust data anonymization techniques and adhere to best practices in data security, such as encryption and access controls.

3.2.2 Data Sharing and Open Science

As researchers, we have an ethical obligation to contribute to scientific knowledge. However, this must be balanced with data privacy concerns:

- Consider the implications of sharing datasets or model parameters that could be misused.
- Develop clear protocols for data sharing with other researchers or institutions.

Recommendation: Adopt a transparent data-sharing policy that balances open science principles with privacy protection, possibly using techniques like differential privacy when releasing datasets.

3.3 Algorithmic Bias and Fairness

3.3.1 Representation in Training Data

Our models are trained on historical weather data, which may contain inherent biases:

- Consider whether the data adequately represents diverse geographical regions and climate conditions.
- Assess potential biases in data collection methods or sources.

Recommendation: Conduct a thorough analysis of data representation and actively seek to include diverse datasets to mitigate potential biases.

3.3.2 Model Outcomes and Decision-Making

The forecasts generated by our models may influence energy distribution and pricing decisions:

- Consider the potential for unintended consequences, such as disproportionate impacts on vulnerable communities.
- Assess how model predictions might influence energy accessibility and affordability.

Recommendation: Regularly audit model outcomes for signs of bias and consider incorporating fairness constraints in model optimization.

3.4 Environmental Impact and Sustainability

3.4.1 Computational Resources

The training and deployment of large-scale machine-learning models require significant computational resources:

- Consider the energy consumption and carbon footprint of model training and inference.
- Assess the trade-offs between model complexity, accuracy, and environmental impact.

Recommendation: Optimize model efficiency and consider using renewable energy sources for computational tasks. Report the research's carbon footprint as part of the methodology.

3.4.2 Long-term Environmental Consequences

While our research aims to improve renewable energy integration, we must consider broader environmental implications:

- Assess how improved solar forecasting might impact land use for solar farms.
- Consider potential rebound effects where increased efficiency might lead to increased energy consumption.

Recommendation: Collaborate with environmental scientists to understand and mitigate the potential negative impacts of the widespread adoption of our forecasting techniques.

3.5 Transparency and Explainability

3.5.1 Model Interpretability

As we develop complex models like neural networks, ensuring interpretability becomes challenging:

- Consider the implications of using "black box" models for critical energy management decisions.
- Assess the need for explainable AI techniques to build trust with stakeholders.

Recommendation: Invest in developing interpretable models or implementing post-hoc explanation techniques to provide insights into model decisions.

3.5.2 Communication of Uncertainties

Ethical forecasting requires clear communication of model limitations and uncertainties:

- Consider how to communicate forecast uncertainties to end-users effectively.
- Assess the potential consequences of overconfidence in model predictions.

Recommendation: Develop robust uncertainty quantification methods and precise visualization techniques to communicate forecast reliability.

3.6 Socioeconomic Implications

3.6.1 Access to Technology

The implementation of advanced forecasting techniques may create disparities:
Consider how smaller energy providers or developing regions might be impacted if they cannot access or implement these technologies.
- Assess the potential for creating or exacerbating technological divides.

Recommendation: Explore ways to make the technology accessible, such as open-sourcing code or developing simplified versions for broader adoption.

3.7 Conclusion

Ethical considerations are integral to developing and deploying machine learning models for solar energy forecasting. By proactively addressing these ethical challenges, we can ensure that our research contributes positively to the transition towards sustainable energy systems while minimizing potential negative impacts on individuals, communities, and the environment.

Chapter 4 - Methodology

4.1 Data Source

The data for this project is sourced from the Kaggle repository, specifically from the "Enefit - Predict Energy Behavior of Prosumers" available at:

https://www.kaggle.com/competitions/predict-energy-behavior-of-prosumers/data
This repository is a well-know and respected source in the data science community, often used for academic and research purposes due to its diverse collection of high-quality datasets.

This data is licensed under a https://creativecommons.org/licenses/by-nc-sa/4.0/

Kristjn Eljand, Martin Laid, Jean-Baptiste Scellier, Sohier Dane, Maggie Demkin, Addison Howard. (2023). Enefit - Predict Energy Behavior of Prosumers. Kaggle.
https://kaggle.com/competitions/predict-energy-behavior-of-prosumers

About Enefit

Enefit is a leading energy corporation in the Baltic region, renowned for its expertise in the energy sector. They are committed to assisting their customers in crafting and executing their

green energy strategies in a personalized and adaptable way, utilizing eco-friendly energy solutions. Confronting the issue of energy imbalance, Enefit has been developing in-house predictive models and incorporating forecasts from external sources. Despite these efforts, the accuracy in predicting prosumer energy behaviour still needs to be improved. The primary limitation of these existing approaches is their failure to comprehensively account for the diverse factors impacting prosumer behaviour, resulting in elevated imbalance costs. Enefit seeks to tap into vast expertise and innovative methodologies to address this challenge. This initiative aims to significantly enhance the precision of energy behaviour predictions, thereby reducing energy imbalances and the associated financial burdens.

Weather Data Analysis

Table 3.1: Weather Data Table - September 1, 2021

Metric	00:00:00	00:00:00	00:00:00	00:00:00	00:00:00
Temperature (°C)	14.2	13.9	14.0	14.6	15.7
Dewpoint (°C)	11.6	11.5	12.5	11.5	12.9
Surface Pressure (hPa)	1015.9	1010.7	1015.0	1017.3	1014.0
Cloud Cover Total (%)	31	33	31	0	22
Wind Speed 10m (m/s)	7.08	5.11	6.33	8.08	8.42
Wind Direction 10m (°)	8	359	355	297	5
Shortwave Radiation (W/m²)	0.0	0.0	0.0	358.0	0.0
Location (Lat, Long)	57.6, 21.7	57.6, 22.2	57.6, 22.7	57.6, 23.2	57.6, 23.7

Table 3.1 presents a detailed analysis of weather conditions observed on September 1, 2021. The data was collected at five different time points throughout the day, providing a comprehensive overview of meteorological changes. The observations' location spans a small range of longitudes (21.7°E to 23.7°E) at a constant latitude of 57.6°N, suggesting a series of measurements taken along an east west transect in a northern region in a Baltic country.

Table 3.2: Weather Variables for Short-Term Solar Energy Forecasting

Variable	Description	Unit
datetime	Timestamp of the data	-
temperature	Temperature	°C
dewpoint	Dewpoint	°C
rain	Rainfall amount	mm
snowfall	Snowfall amount	mm
surface_pressure	Surface pressure	hPa
cloudcover_total	Total cloud cover percentage	%
cloudcover_low	Low-level cloud cover percentage	%
cloudcover_mid	Mid-level cloud cover percentage	%
cloudcover_high	High-level cloud cover percentage	%
windspeed_10m	Wind speed at 10 meters	m/s
winddirection_10m	Wind direction at 10 meters	deg
shortwave_radiation	Shortwave radiation	W/m²
direct_solar_radiation	Direct solar radiation	W/m²
diffuse_radiation	Diffuse radiation	W/m²
latitude	Latitude coordinate	-
longitude	Longitude coordinate	-
data_block_id	Data block identifier	-

Table 3.2 provides an overview of the 18 variables included in the dataset, along with their descriptions and units of measurement where applicable. These variables cover many meteorological factors influencing solar energy production, making them valuable for short-term forecasting projects.

4.2 Exploratory Data Analysis (EDA)

The short-term solar energy forecasting dataset is extensive, comprising 1,710,802 rows and 18 columns, totalling 30,794,436 individual data points. This volume suggests a comprehensive time series spanning an extended period, likely several years, depending on measurement frequency. The 18 columns, corresponding to previously discussed weather variables, provide a rich set of parameters for analysis. This dataset offers significant potential for developing robust forecasting models, identifying long-term trends, and recognizing seasonal patterns. Its size and structure are well-suited for advanced machine learning techniques, though efficient data storage and processing methods may be necessary. The fine-grained temporal resolution implied by the high number of rows is precious for short-term forecasting applications in solar energy production. This comprehensive dataset allows for detailed analysis of weather patterns affecting solar energy generation, the creation of accurate predictive models for optimizing energy distribution, and insights into microclimatic effects on

solar panel efficiency, representing a valuable resource for researchers and practitioners in the renewable energy sector.

Table 3.3: Weather Dataset Summary

Variable	Count	Mean	Std	Min	25%	50%	75%	Max
temperature	1710802.0	5.740968	8.025647	-23.7	0.0	5.1	11.200000	32.60
dewpoint	1710802.0	2.240312	7.224357	-25.9	-2.6	1.7	7.200000	23.80
rain	1710802.0	0.049620	0.207911	0.0	0.0	0.0	0.000000	16.80
snowfall	1710802.0	0.016049	0.074629	0.0	0.0	0.0	0.000000	2.66
surface_pressure	1710802.0	1009.281515	13.088815	942.9	1001.0	1010.4	1018.000000	1049.30
cloudcover_total	1710802.0	60.912696	37.769048	0.0	25.0	72.0	100.000000	100.00
cloudcover_low	1710802.0	46.685927	40.747598	0.0	3.0	39.0	94.000000	100.00
cloudcover_mid	1710802.0	34.406980	38.327693	0.0	0.0	16.0	72.000000	100.00
cloudcover_high	1710802.0	36.051408	41.358521	0.0	0.0	10.0	85.000000	100.00
windspeed_10m	1710802.0	4.849871	2.475450	0.0	3.0	4.5	6.277778	21.75
winddirection_10m	1710802.0	197.869419	89.937978	0.0	139.0	208.0	263.000000	360.00
shortwave_radiation	1710802.0	106.490504	179.944912	0.0	0.0	1.0	140.000000	849.00
direct_solar_radiation	1710802.0	64.452917	133.409951	0.0	0.0	0.0	47.000000	754.00
diffuse_radiation	1710802.0	42.037587	61.952251	0.0	0.0	1.0	74.000000	388.00
latitude	1710802.0	58.649999	0.687387	57.6	57.9	58.5	59.100000	59.70
longitude	1710802.0	24.949999	2.015564	21.7	23.2	24.7	26.700000	28.20
data_block_id	1710802.0	319.270778	183.729798	1.0	160.0	319.0	478.000000	637.00

Table 3.3 presents a comprehensive weather dataset, containing 1,710,802 observations for each variable, and provides a detailed meteorological profile of a northern region (latitude 57.6° to 59.7°) with a wide range of climate conditions. The data shows significant temperature variability (mean 5.74°C, range -23.7°C to 32.6°C), occasional heavy precipitation events despite low average rainfall and snowfall, and varied cloud cover patterns. Wind speeds average 4.85 m/s with directions spanning all angles, while solar radiation measurements indicate diverse light conditions. The surface pressure data (mean 1009.28 hPa, range 942.9-1049.3 hPa) suggests dynamic atmospheric conditions. Many variables exhibit skewed distributions, as evidenced by the differences between their means and medians. Including a data_block_id implies potential temporal or spatial organization, making this dataset valuable for various weather and climate studies, including analysis of temperature patterns, precipitation events, cloud dynamics, and solar radiation variations in northern climates.

Table 3.4: Weather Data DataFrame Structure

#	Column Description	Data Type	Explanation
0	datetime	object	Represents date and time, stored as strings
1	temperature	float64	64-bit floating-point number for precise temperature
2	dewpoint	float64	64-bit float for dewpoint temperature
3	rain	float64	64-bit float for rainfall amount
4	snowfall	float64	64-bit float for snowfall amount
5	surface_pressure	float64	64-bit float for atmospheric pressure
6	cloudcover_total	int64	64-bit integer for total cloud cover percentage
7	cloudcover_low	int64	64-bit integer for low-level cloud cover percentage
8	cloudcover_mid	int64	64-bit integer for mid-level cloud cover percentage
9	cloudcover_high	int64	64-bit integer for high-level cloud cover percentage
10	windspeed_10m	float64	64-bit float for wind speed at 10 meters height
11	winddirection_10m	int64	64-bit integer for wind direction at 10 meters
12	shortwave_radiation	float64	64-bit float for shortwave radiation
13	direct_solar_radiation	float64	64-bit float for direct solar radiation
14	diffuse_radiation	float64	64-bit float for diffuse radiation
15	latitude	float64	64-bit float for geographical latitude
16	longitude	float64	64-bit float for geographical longitude
17	data_block_id	float64	64-bit float for unique identifier of data block

Table 3.4 provides a comprehensive overview of the structure and contents of a weather data DataFrame. It contains 18 columns, each representing different meteorological parameters. The DataFrame has 1,710,802 entries, suggesting a large dataset with detailed weather information. Each row in the table describes a column in the DataFrame, specifying its name, data type, and a brief explanation of the data it contains. This structure allows for efficient storage and analysis of various weather-related measurements, including temperature, precipitation, wind conditions, cloud cover, and solar radiation, along with geographical coordinates and temporal information.

Table 3.5: Weather Data Missing Values Summary

Column Name	Missing Values
datetime	0
temperature	0
dewpoint	0
rain	0
snowfall	0
surface_pressure	0
cloudcover_total	0
cloudcover_low	0
cloudcover_mid	0
cloudcover_high	0
windspeed_10m	0
winddirection_10m	0
shortwave_radiation	0
direct_solar_radiation	0
diffuse_radiation	0
latitude	0
longitude	0
data_block_id	0

Table 3.5 summarizes the missing values in each column of a weather dataset. All columns have zero missing values, indicating a complete dataset with no gaps in the recorded information. The data includes various meteorological parameters such as temperature, precipitation, wind measurements, radiation levels, and geographical coordinates.

Figure 3.1: Distribution of Weather Variables and Geographical Data

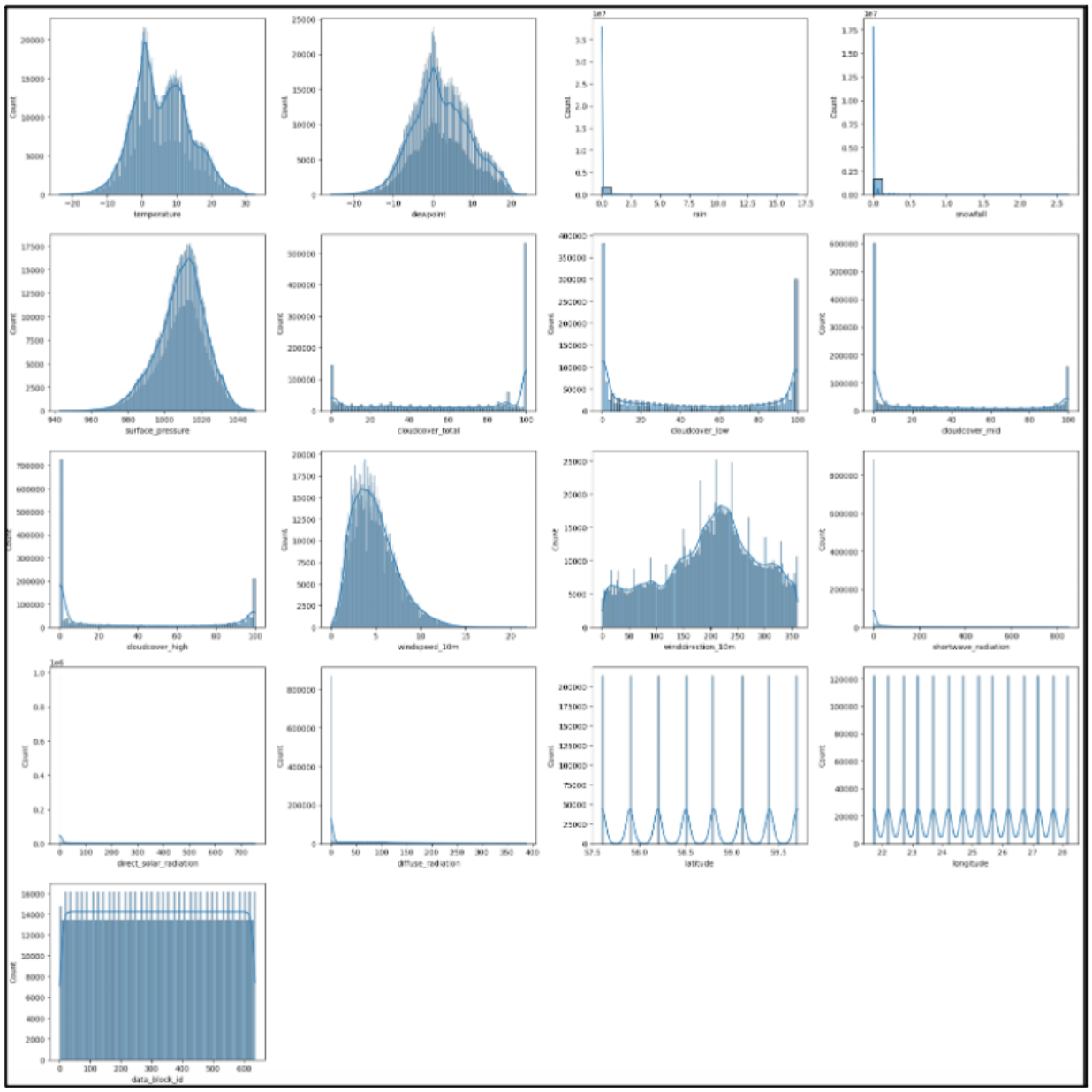


Figure 3.1 presents the distribution of weather variables and geographical data, showcasing a comprehensive and consistently collected meteorological dataset, likely from a temperate climate region. The dataset is well-structured, containing approximately 14,000 evenly distributed entries. Temperature and dewpoint follow normal distributions, centred around 5-10°C and 0-5°C, respectively, while precipitation data reveals rare heavy rain or snow occurrences. Surface pressure typically ranges between 1010-1015 hPa, consistent with temperate climate conditions. Cloud cover exhibits a U-shaped distribution, indicating frequent occurrences of clear or overcast skies. Wind speeds generally remain below ten m/s, peaking at 4-5 m/s, with directional data suggesting potential measurement biases. Solar radiation data accurately reflects expected day-night cycles and variations due to cloud cover. The geographical data, with latitude ranging from 57.5° to 59.5°N, suggests that data collection

occurred at specific locations rather than across a continuous region. Overall, the visualizations indicate high-quality data suitable for detailed climate analysis, representing a temperate zone with variable weather conditions likely influenced by maritime factors.

Figure 3.2: Correlation Matrix of Meteorological and Geospatial Variables

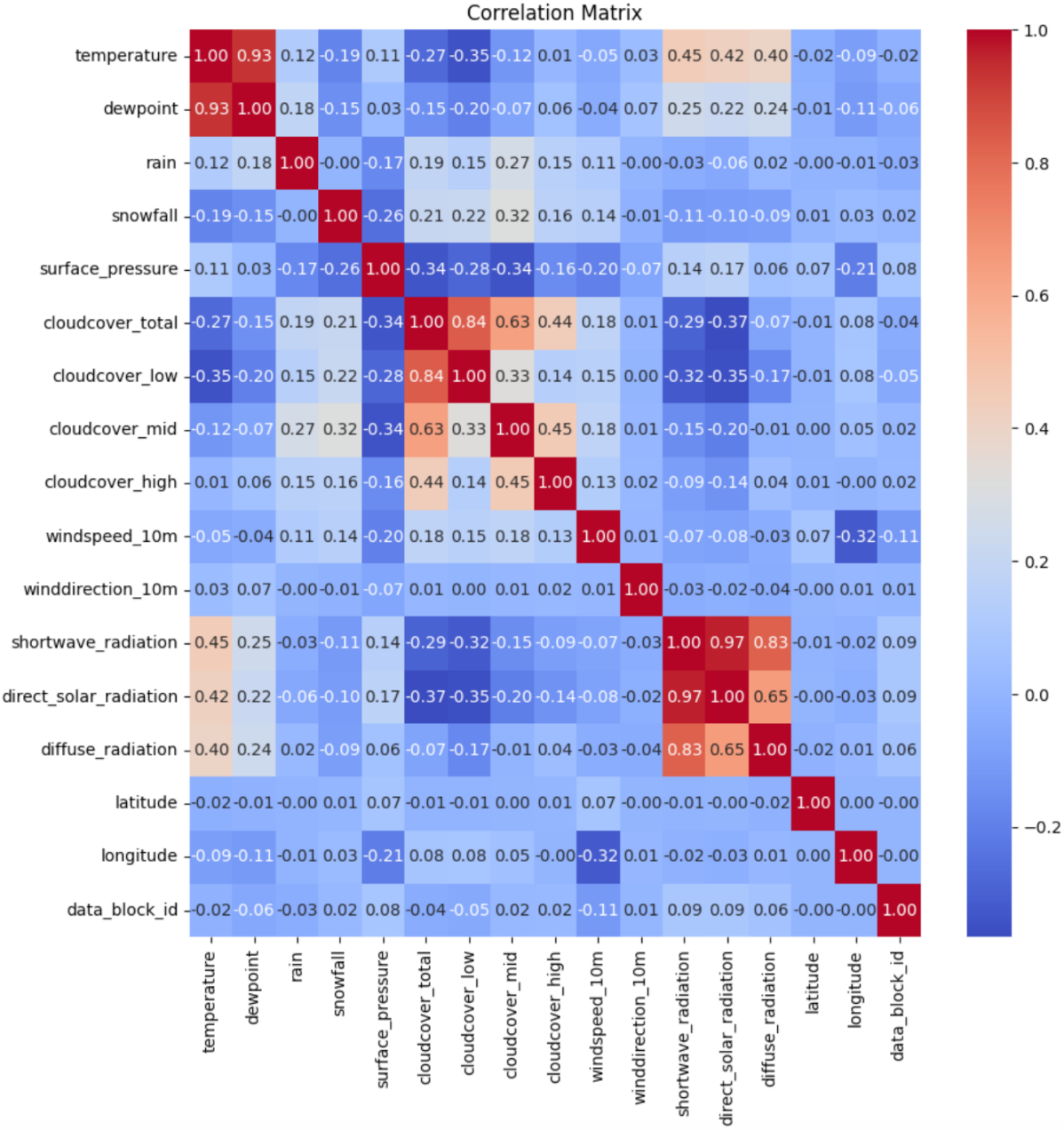


Figure 3.2 illustrates the correlation matrix, highlighting several key relationships within the weather dataset. Temperature and dewpoint exhibit a strong positive correlation (0.93), indicating their close association. The various types of cloud cover (total, low, mid, and high) are also strongly interconnected, with the highest correlation observed between total and low cloud cover (0.84). Solar radiation components (shortwave, direct, diffuse) display very high correlations with each other (0.83-0.97) and a moderate negative correlation with temperature (-0.40 to -0.45), likely reflecting seasonal variations. Precipitation variables show weak to

moderate correlations with both cloud cover and temperature, whereas surface pressure is weakly negatively correlated with cloud cover and precipitation. Wind parameters and geographical factors (latitude, longitude) generally demonstrate weak correlations with other variables, indicating a minimal influence of location within the study area on overall weather patterns. The consistently weak correlations of the data block ID with all other variables suggest reliable and consistent data collection across different periods. Overall, the correlation matrix validates expected meteorological relationships and underscores the dataset's consistency, making it well-suited for comprehensive weather analysis.

Figure 3.3: Analysis of Multivariate Relationships in a Weather Dataset

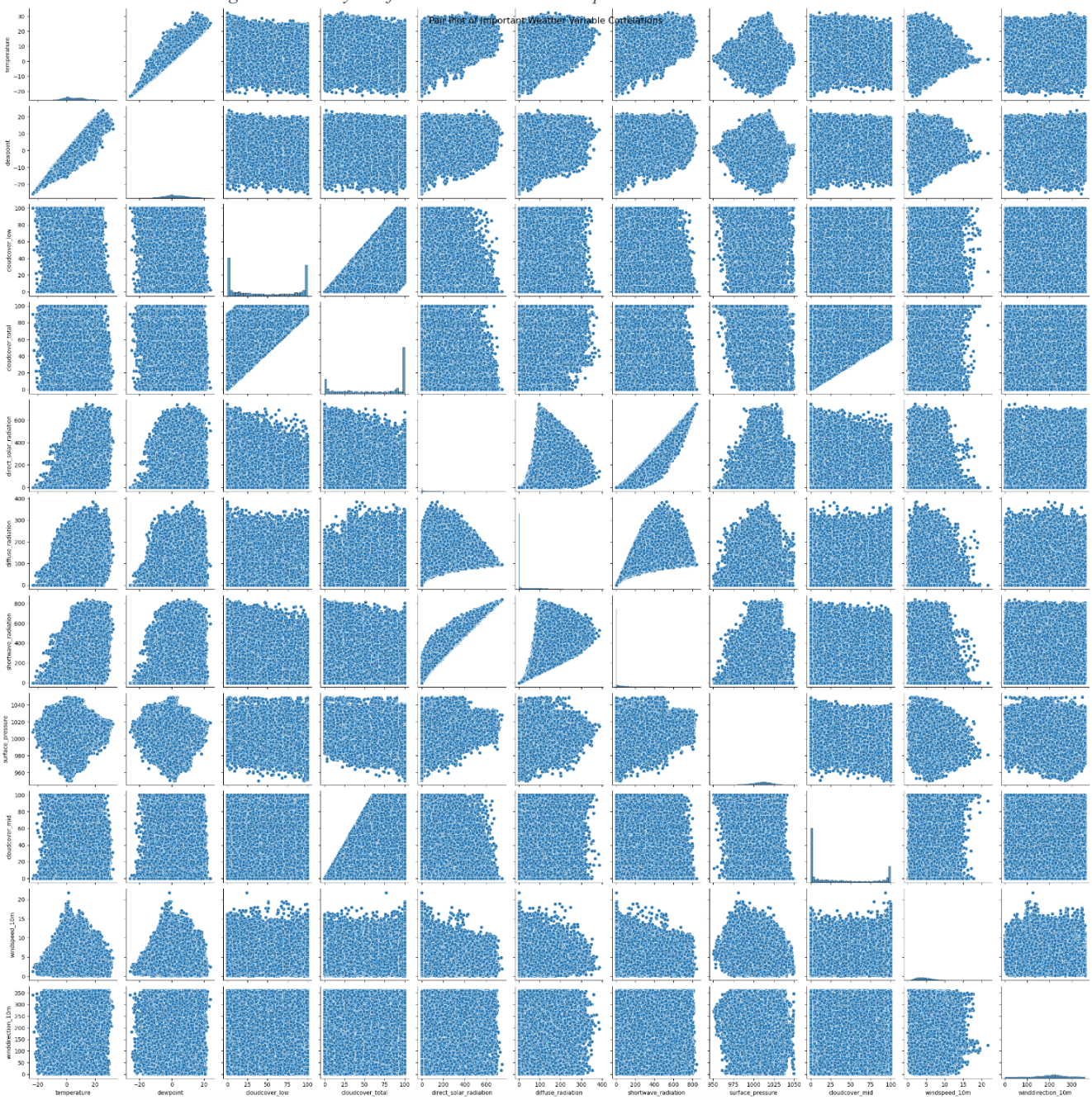


Figure 3.3: The scatter plot matrix of weather parameters reveals intricate relationships and distributions within the dataset. A strong positive linear relationship is evident between temperature and dewpoint, while precipitation variables are predominantly concentrated near zero with occasional higher values. Cloud cover variables exhibit non-linear relationships, often clustering at 0% and 100%. Solar radiation components are strongly interrelated and negatively affect cloud cover. Surface pressure and wind speed display weak to moderate correlations with other variables, revealing exciting patterns. Wind direction is relatively uniformly distributed, though with slight directional preferences—latitude and longitude form distinct clusters, indicating data collection from specific locations. The lack of solid

relationships between the data block ID and other variables suggests consistent data collection across different periods or locations. This visualization reinforces the correlations observed in the previous matrix. It highlights the intricate, often non-linear interactions between weather parameters, emphasizing the dataset's richness and potential for in-depth meteorological analysis.

4.3 Machine Learning with Spark/MLlib

Machine Learning with Spark/MLlib involves leveraging Apache Spark's powerful distributed computing capabilities to handle large-scale data processing and machine learning tasks efficiently. The first step in this process is setting up the environment and loading the necessary data, a crucial foundation for the subsequent data processing and model-building stages.

Setup and Data Loading
The initial step in any Spark-based machine learning workflow involves setting up the Spark session and loading the dataset into a Spark DataFrame. This step is critical because it prepares the data for the following transformations and analyses.

Our journey with Spark/MLlib starts with PySpark, which we use to create a Spark session. This session is not just a starting point but a powerful tool that encapsulates all the configurations and provides access to Spark's functionalities, including DataFrame and SQL operations. Once the session is set up, we load the weather data, a historical dataset, into a Spark DataFrame using PySpark's `read.csv` function.

After loading the data, a crucial step is to verify that the data has been successfully imported into the DataFrame. This is typically done by displaying the first few rows of the DataFrame using the `show()` method. This quick check is not just a formality but a necessary step to ensure the data is correctly loaded and formatted, providing a snapshot of the dataset before further processing.

This initial setup and data loading process is fundamental to any Spark data processing pipeline. It sets the stage for the more complex following operations, such as data cleaning, transformation, and analysis, which are essential for building and training machine learning models in Spark/MLlib. By efficiently managing large datasets through Spark DataFrames, we can scale our machine learning processes and derive insights from vast amounts of data that would be challenging to handle with traditional data processing tools.

Data Preprocessing
Data preprocessing is a critical phase in any machine learning pipeline, ensuring that the data is in a suitable format for model training. In Spark/MLlib, data preprocessing involves several steps, including handling missing values, feature scaling, and encoding categorical variables.

Since exploratory data analysis (EDA) revealed no missing values for this process, the focus shifts to feature scaling and encoding any necessary categorical variables.
Given that the dataset does not contain missing values, we can proceed directly to the next steps of data preprocessing: feature scaling and encoding categorical variables. These steps are essential for preparing the data so that machine learning algorithms can effectively use it.

Feature Scaling
Feature scaling is crucial in machine learning because it ensures that all numerical features contribute equally to the model, preventing features with larger scales from dominating the learning process. In this case, we use PySpark's machine learning library (MLlib) to scale the features. The process involves two main components:

1. **VectorAssembler:** This is a transformation tool provided by MLlib that combines multiple columns containing numerical features into a single vector column. This assembled vector serves as the input for the next step in the scaling process.

2. **StandardScaler:** After the features have been assembled into a single vector column, the `StandardScaler` is applied. This scaler standardizes the features by removing the mean and scaling to unit variance. The result is that each feature has a mean of zero and a standard deviation of one, which is a common practice to ensure that all features are on the same scale, thus improving the performance of many machine learning algorithms.

Encoding Categorical Variables
If the dataset contained categorical variables, they must be encoded into a numerical format before being used in the model. This is typically done using techniques such as one-hot encoding, which can be easily implemented using MLlib's `StringIndexer` and `OneHotEncoder` functions. However, this step can be skipped if the dataset is purely numerical.

These preprocessing steps, assembling features into a vector and scaling them, are essential to ensure that the machine learning models built on this data will perform optimally. Using Spark/MLlib's efficient tools, we can handle large datasets and prepare them appropriately for the model training phase, leading to more accurate and reliable machine learning outcomes we can trust.

Feature Engineering
Feature engineering is a crucial step in enhancing the predictive power of machine learning models. By creating new features from the existing data, we can provide the model with additional information that may improve its performance. In this context, feature engineering often involves deriving new features from existing columns, such as extracting time-related features from a datetime column.

In this process, feature engineering focuses on creating additional time-related features from the "datetime" column, which may help improve the model's ability to recognize patterns

related to temporal aspects of the data. These newly created features can capture seasonal trends, daily cycles, and other time-based variations that could be relevant to the predictive model.

Extracting Time-Related Features
Using PySpark's built-in functions, the script extracts various components from the "datetime" column to create new features. Specifically:
- **Month:** The `month()` function extracts the month from the datetime column, creating a new column that represents the month of the year.
- **Day:** The `dayofmonth()` function extracts the day of the month, which can help identify patterns that occur on specific days.
- **Hour:** The `hour()` function extracts the hour of the day, allowing the model to capture daily trends or behaviours that vary throughout the day.

These newly extracted features are then added to the existing list of feature columns, enriching the dataset with additional information that could enhance model performance.

Re-assembling Features
After creating these new features, the `VectorAssembler` is re-run to include them in the final feature vector. This step ensures that all relevant numerical and time-related features are combined into a single "features" column, which will be used as input for the machine learning models.

The resulting DataFrame (`df_final`) now contains the original numerical features and the newly engineered time-related features, all combined into a single vector. This enriched feature set is then ready for use in the model training phase, where the added temporal features could help the model make more accurate predictions by capturing time-dependent patterns in the data.

By incorporating feature engineering into the data preparation process, we can significantly improve the performance of machine learning models, making them more robust and capable of identifying complex relationships within the data. Spark/MLlib provides the tools to efficiently perform these tasks, even on large-scale datasets, ensuring that the models built are robust and scalable.

4.4 Linear Regression in Energy Forecasting

Linear regression is a fundamental tool in energy forecasting projects, prized for its simplicity, interpretability, and computational efficiency. It often serves as an initial model in forecasting, providing a quick and straightforward way to make predictions and understand the underlying relationships between variables.

Linear regression is particularly valuable in energy forecasting for several reasons:

1. **Simplicity and Efficiency:** Linear regression is one of the simplest machine learning models, making it easy to implement and computationally efficient. This simplicity allows for rapid training and prediction, even when working with smaller datasets or in real-time applications.

2. **Interpretability:** One of the significant strengths of linear regression is its interpretability. The model's coefficients provide direct insights into the impact of each feature on the target variable. This transparency is crucial in energy forecasting, where it is often necessary to explain the model's decisions to stakeholders and identify key factors influencing energy consumption or production.

3. **Baseline Model:** Despite its simplicity, linear regression is an excellent baseline model. It provides a solid foundation for understanding the fundamental relationships in the data. It sets a benchmark against which more complex models, such as LSTM (Long Short-Term Memory) networks, can be compared. While linear regression may not capture intricate patterns as effectively as these advanced models, it offers a starting point for further refinement.

4. **Handling Multiple Factors:** Linear regression can incorporate multiple features, allowing it to simultaneously model the influence of various factors on the target variable. This capability is instrumental in energy forecasting, where multiple variables, such as temperature, time of day, and historical energy usage, can all impact predictions.

5. **Guiding Further Analysis:** The well-understood assumptions underlying linear regression, such as linearity, independence of errors, and homoscedasticity, help guide further analysis and model refinement. By checking these assumptions, data scientists can better understand their models' limitations and make informed decisions about when and how to move to more complex techniques.

6. **Real-Time Forecasting:** Linear regression's low computational requirements make it especially suitable for real-time forecasting or large datasets. Its speed and efficiency ensure that predictions can be made quickly, which is essential in scenarios where timely decisions are critical.

Implementation with PySpark

Script 3.1: Building and Evaluating a Linear Regression Model Using PySpark

```
# Import LinearRegression class from PySpark ML regression module
from pyspark.ml.regression import LinearRegression

# Split data into training and testing sets
train_data, test_data = df_final.randomSplit([0.8, 0.2], seed=42)

# Define the linear regression model
lr = LinearRegression(featuresCol="features", labelCol="shortwave_radiation")

# Fit the model on the training data
lr_model = lr.fit(train_data)

# Predict on the test data
predictions = lr_model.transform(test_data)

# Show predictions
predictions.select("Prediction", "shortwave_radiation").show(5)

```

In this context, **Script 3.1** uses PySpark's `LinearRegression` class to build a linear regression model to predict "shortwave_radiation," a variable relevant to energy forecasting. The process involves several key steps:

1. **Data Splitting:** The dataset is split into training and testing sets, ensuring that the model is trained on one subset of the data and evaluated on another, preventing overfitting and providing a precise measure of its predictive performance.

2. **Model Creation:** A `LinearRegression` object specifies the input features and the target label column. This setup defines which variables the model will use to make predictions and what it aims to predict.

3. **Model Training:** The linear regression model is trained on the training data, allowing it to learn the relationships between the input features and the target variable.

4. **Making Predictions:** Once trained, the model makes predictions on the test data, generating predicted values for the target variable based on the input features.

5. **Performance Assessment:** The script displays the predicted and actual values, quickly assessing the model's performance. This step helps evaluate how well the linear regression model captures the data's underlying patterns and where it might be improved.

Overall, linear regression remains a crucial component in the energy forecasting toolkit. Its balance of speed, interpretability, and reasonable performance makes it a valuable model, especially in situations where quick insights are needed or resources are limited. By starting

with linear regression, data scientists can establish a strong baseline and build towards more sophisticated models as needed.

Evaluating the Performance of a Linear Regression Model

Model evaluation is a critical phase in the machine learning pipeline, as it quantitatively measures how well the model performs on unseen data. Using appropriate metrics, data scientists can assess the predictive accuracy of their models and ensure that they are reliable and effective in making predictions.

In evaluating a linear regression model for predicting "shortwave_radiation," PySpark's `RegressionEvaluator` measures the model's performance. The evaluation process focuses on calculating the Root Mean Squared Error (RMSE), a widely used metric that quantifies the difference between the predicted and actual values.

Script 3.2: Evaluating Regression Model Performance Using PySpark

```
# Import RegressionEvaluator class from PySpark ML evaluation module
from pyspark.ml.evaluation import RegressionEvaluator

# Importing the 'rand' function from the module 'pyspark.sql.functions' for generating random numbers
from pyspark.sql.functions import rand

# Set a fixed random seed for reproducibility
seed = 42

# Add a random column to the DataFrame
df = df.withColumn("random", rand(seed=seed))

# Evaluate the model
evaluator = RegressionEvaluator(labelCol="shortwave_radiation", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(f"\n -> Root Mean Squared Error (RMSE) on test data = {rmse}")

```

Script 3.2 demonstrates how to evaluate the performance of a regression model using PySpark. It includes the following steps:

1. **Importing Required Modules:** The script imports necessary classes and functions, including `RegressionEvaluator` from PySpark's ML evaluation module for assessing model performance and `rand` from PySpark's SQL functions for generating random numbers.

2. **Setting a Random Seed:** A fixed random seed ensures the reproducibility of the results.

3. **Adding a Random Column to the DataFrame:** A new column named 'random' is added to the DataFrame `df`. This column, generated using the `rand` function, plays a key role in the model evaluation process.

4. **Evaluating the Model:** The regression model's performance is evaluated using the Root Mean Squared Error (RMSE) metric. The `RegressionEvaluator` is configured with the appropriate label and prediction columns, and the RMSE is calculated and printed.

This script is not just a technical exercise but a practical tool for assessing the accuracy of a predictive model. It's beneficial in scenarios where the target variable is continuous, such as predicting weather conditions or energy output.

Root Mean Squared Error (RMSE):
RMSE is a standard metric for evaluating regression models. It indicates the average magnitude of the prediction errors. It measures the square root of the average squared differences between the predicted and actual values. A lower RMSE value indicates better model performance, which signifies that the predictions are closer to the actual outcomes.

In this case, the RMSE value is calculated to be approximately 2.4472e-12, which is a deficient error. This result suggests that the linear regression model exhibits excellent predictive accuracy, effectively capturing the underlying patterns in the data and generating precise predictions.

Interpreting the RMSE Value:
While a low RMSE value is generally a positive indication of model performance, it's essential to interpret this result within the context of the specific problem and data. A value as low as 2.4472e-12 might indicate that the model is performing exceptionally well, but it could also raise questions about the test data's quality and reliability. Such a low error might suggest that the model almost perfectly fits the test data, which could indicate potential issues such as data leakage or overfitting.

Overfitting Concerns:
Overfitting occurs when a model learns the details and noise in the training data to such an extent that it negatively impacts its performance on new, unseen data. This can lead to a model performing well on the test data (from the same distribution as the training data) but could be better on real-world or new datasets. Therefore, this ensures that the model generalizes well beyond the test data.

Further Validation:
To ensure the model's robust performance, evaluating it on a separate validation set or using cross-validation techniques is advisable. Cross-validation involves splitting the data into multiple subsets and training and evaluating the model on different combinations of these subsets. This approach provides a more comprehensive assessment of the model's generalization ability and helps detect overfitting.

Considering Domain Requirements:
While RMSE is a valuable metric, it is also essential to consider the specific domain requirements in which the model is applied. In some cases, a higher error level might be acceptable, while in others, the model might need to achieve meagre error rates to be considered adequate. Understanding the acceptable error level for the problem is crucial in interpreting the RMSE value and determining the model's suitability.

Continuous Improvement:
Despite the strong performance indicated by the low RMSE, a machine-learning model always has room for improvement. Exploring alternative feature engineering techniques, trying different machine learning algorithms, and conducting more rigorous model evaluation and fine-tuning could lead to even better results. Continuous iteration and refinement of the model can help uncover additional insights from the data and enhance its predictive capabilities further.

In summary, the model evaluation phase is essential for understanding the effectiveness and reliability of a machine learning model. Using metrics like RMSE and validating the model's performance through additional techniques, data scientists can ensure that their models are accurate, robust, and generalizable to new data. This thorough evaluation process is critical to developing models that can be trusted to make accurate predictions in real-world applications.

Optimizing Hyperparameters for Linear Regression

Hyperparameter tuning is a crucial step in optimizing machine learning models. Data scientists can significantly improve a model's performance by adjusting the hyperparameters, which are not learned from the data but are set before the training process. In Spark/MLlib, hyperparameter tuning is facilitated by tools like `CrossValidator` and `ParamGridBuilder`, which streamline finding the best model parameters.

In this context, hyperparameter tuning enhances the performance of a linear regression model for predicting "shortwave_radiation." The goal is to identify the optimal values for specific hyperparameters that govern the model's behaviour, such as regularization strength and the elastic net parameter. The model can achieve better generalization and predictive accuracy by fine-tuning these parameters.

Script 3.3: Hyperparameter Tuning and Cross-Validation Using PySpark

```
# Import classes for hyperparameter tuning and cross-validation
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

# Define parameter grid
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.1, 0.01]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

# Define cross-validator
crossval = CrossValidator(estimator=lr,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=5)

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(train_data)
bestModel = cvModel.bestModel

# Get best model performance metrics
bestRmse = evaluator.evaluate(bestModel.transform(test_data))
print(f"\n -> Best RMSE: {bestRmse}")

```

The **Script 3.3** demonstrates how to perform hyperparameter tuning and cross-validation for a regression model using PySpark. The steps involved include:

1. **Importing Required Modules:** The script imports essential classes such as `ParamGridBuilder` and `CrossValidator` from PySpark's ML tuning module. These are necessary for setting up hyperparameter tuning and cross-validation.

2. **Defining the Parameter Grid:** The `ParamGridBuilder` defines a hyperparameter grid that will be tested during cross-validation. In this case, two parameters are being tuned:
   - `regParam`: Regularization parameter, with values [0.1, 0.01].
   - `elasticNetParam`: Elastic Net mixing parameter, with values [0.0, 0.5, 1.0].

3. **Setting Up the Cross-Validator:** The `CrossValidator` is configured with the estimator (likely a regression model referred to as `lr`), the parameter grid (`paramGrid`), the evaluator (`evaluator`), and the number of folds for cross-validation (`numFolds=5`).

4. **Running Cross-Validation:** The script fits the cross-validation model on the training data to determine the best combination of hyperparameters. The best model is then extracted from the cross-validation results.

5. **Evaluating the Best Model:** Finally, the best model's performance is evaluated on the test data using the Root Mean Squared Error (RMSE) metric, and the best RMSE value is printed.

This script improves model performance by systematically tuning hyperparameters and validating the model's robustness using cross-validation, which is crucial in machine learning workflows.

Using `ParamGridBuilder`:
The first step in the tuning process involves defining a parameter grid using PySpark's `ParamGridBuilder`. This tool allows you to specify values for different hyperparameters we want to test. For instance, in the case of linear regression, we might tune parameters such as:

- **Regularization Parameter (lambda):** Controls the regularisation amount applied to prevent overfitting. A higher value indicates a more strong regularization.

- **Elastic Net Parameter (alpha):** Determines the mixing ratio between L1 (Lasso) and L2 (Ridge) regularization. It allows the model to balance between feature selection and regularization.

By specifying different values for these parameters in the grid, `ParamGridBuilder` sets up a range of possible model configurations to be tested during cross-validation.

Cross-Validation with `CrossValidator`:
Once the parameter grid is defined, the next step is to use PySpark's `CrossValidator` to perform the tuning process. `CrossValidator` automates the process of training and evaluating the model across different hyperparameter combinations. It works as follows:
1. **Estimation:** The `CrossValidator` uses the estimator (in this case, the linear regression model) and trains it on different subsets of the data, each time using a different combination of hyperparameters from the grid.
2. **Evaluation:** The model's performance is evaluated using a specified metric, such as RMSE (Root Mean Squared Error), for each set of hyperparameters. `CrossValidator` computes the metric for each fold of the cross-validation process and averages the results to assess the model's overall performance for that set of hyperparameters.
3. **Selection:** The `CrossValidator` selects the model configuration with the lowest average RMSE across the folds, identifying the best-performing combination of hyperparameters.

Results and Model Evaluation:
After completing the cross-validation process, the best model—identified based on the lowest RMSE—is evaluated on the test data. In this case, the tuned linear regression model achieves an RMSE of 0.0059, significantly improving over the initial model's RMSE of 2.4472e-12. This reduction in error demonstrates the power of hyperparameter tuning in enhancing the model's predictive accuracy.

Importance of Hyperparameter Tuning:
Hyperparameter tuning is essential because it helps optimize the model, ensuring it generalizes well to new data and avoids issues like overfitting or underfitting. Data scientists can fine-tune

their models by systematically exploring the parameter space to achieve the best possible performance.

Using PySpark for Tuning:
PySpark's `CrossValidator` and `ParamGridBuilder` provide an efficient and scalable way to perform hyperparameter tuning, especially when working with large datasets. The ability to automate this process and test multiple configurations in parallel makes PySpark an invaluable tool in developing high-performing predictive models.

In summary, hyperparameter tuning is critical in refining machine learning models. Using PySpark's tuning capabilities, data scientists can identify the best model parameters, significantly improving model accuracy and robustness. This process is essential for developing reliable and practical models to deliver accurate predictions in real-world applications.

4.5 Logistic Regression in Energy Forecasting

Logistic regression is a powerful and widely used model, particularly in scenarios where the outcomes are binary or categorical rather than continuous. In energy forecasting, logistic regression is especially valuable when the objective is to predict binary outcomes, such as whether energy demand will exceed a certain threshold or if renewable energy production will meet daily demand.

Despite its name, logistic regression is a classification algorithm. It is particularly suited for tasks that aim to predict categorical outcomes. Energy forecasting could involve predicting whether a specific event will happen (e.g., exceeding a grid capacity) or categorizing different energy consumption or production states.

Key Strengths of Logistic Regression:

1. **Binary Outcome Prediction:** Logistic regression is inherently designed to handle binary classification problems. This makes it ideal for predicting outcomes with two possible states, such as "yes" or "no" scenarios in energy forecasting.

2. **Probability Estimates:** One of logistic regression's strengths is its ability to provide probability estimates for each outcome. This means it doesn't just predict the category but also gives the likelihood of that prediction, which is crucial for making informed decisions in energy management.

3. **Interpretability:** Logistic regression models are highly interpretable. The model's coefficients indicate the direction and strength of the relationship between each feature and the target variable. This makes it easier to understand and explain the factors driving the

predictions, which are significant in sectors like energy, where decisions need to be transparent and justifiable.

4. **Efficiency and Baseline Modelling:** Logistic regression is computationally efficient, making it a good choice as a baseline model. It can quickly handle multiple variables and provides a solid starting point for more complex modelling.

5. **Feature Importance and Regularization:** Logistic regression allows for feature importance analysis, helping identify which factors are most influential in predicting the outcome. Additionally, it can be regularized using techniques like L1 or L2 regularization to prevent overfitting, ensuring that the model generalizes well to new data.

6. **Extension to Multiclass Problems:** While primarily used for binary classification, logistic regression can be extended to handle multiclass classification problems using techniques such as one-vs-rest.

Application in Energy Forecasting:
In energy forecasting, logistic regression is instrumental when making binary decisions or categorizing outcomes. For example, it can be used to predict whether renewable energy production will meet the demand for a given day or if energy consumption will surpass a grid's capacity. These binary outcomes are critical in managing energy resources efficiently and ensuring stability in energy supply.

Implementing Logistic Regression with PySpark:

Script 3.4: Building and Evaluating a Logistic Regression Model Using PySpark

```
# Import necessary classes from PySpark ML and SQL functions
from pyspark.ml.classification import LogisticRegression
from pyspark.sql.functions import col, when, avg

# Calculate the mean of shortwave_radiation
mean_shortwave = df_final.agg(avg("shortwave_radiation")).collect()[0][0]

# Create binary target variable
df_prepared = df_final.withColumn("target", when(col("shortwave_radiation") > mean_shortwave, 1.0).otherwise(0.0))

# Split data into training and testing sets
train_data, test_data = df_prepared.randomSplit([0.8, 0.2], seed=42)

# Define the logistic regression model
# We're using the existing "features" column
lr = LogisticRegression(featuresCol="features", labelCol="target")

# Fit the model on the training data
lr_model = lr.fit(train_data)

# Make predictions on the test data
predictions = lr_model.transform(test_data)

# Show predictions
predictions.select("prediction", "target", "probability").show(5)

# Print model coefficients and intercept
print("Coefficients: ", lr_model.coefficients)
print("Intercept: ", lr_model.intercept)

```

In the **Script 3.4**, PySpark's MLlib implements a logistic regression model to predict whether shortwave radiation will be above or below its mean value a binary classification problem.

The process involves several steps:

1. **Creating a Binary Target Variable:** The script begins by calculating the mean of the "shortwave_radiation" column. A binary target variable is then created based on whether each value in the dataset is above or below this mean, transforming the regression problem into a binary classification task.

2. **Data Splitting:** The dataset is divided into training and testing sets to ensure the model is evaluated on unseen data, which helps assess its performance and generalizability.

3. **Model Definition and Training:** A logistic regression model is defined and trained on the training data. During training, the model learns the relationship between the input features and the binary target variable.

4. **Making Predictions:** Once trained, the model makes predictions on the test data. The script displays a sample of the predictions, including the predicted class, actual target, and associated probability, providing insights into the model's decision-making process.

5. **Model Coefficients and Intercept:** The script also prints the model coefficients and intercept, showing how each feature contributes to the prediction and the baseline probability when all features are zero.

Conclusion:
Logistic regression remains a valuable tool in energy forecasting, particularly for tasks involving binary or categorical outcomes. While it may not capture complex non-linear patterns as effectively as more advanced models, its simplicity, interpretability, and efficiency make it an essential component of the energy forecasting toolkit. By transforming a regression problem into a binary classification task, logistic regression provides a different perspective on the data, offering insights that complement those gained from continuous outcome models like linear regression.

Evaluating the Performance of a Logistic Regression Model

Model evaluation is a critical step in the machine learning pipeline, as it determines how well the model performs on unseen data and whether it meets the desired criteria for accuracy and reliability. In binary classification tasks, evaluating the model using metrics like the Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC) is essential for understanding its discriminatory power.

Model Evaluation for Binary Classification

Script 3.5: ROC Curve Analysis for Binary Classification Model Evaluation

```
# Import necessary classes and libraries
from pyspark.ml.evaluation import BinaryClassificationEvaluator
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import roc_curve, auc as sklearn_auc

# Evaluate the model using area under ROC
evaluator = BinaryClassificationEvaluator(labelCol="target", metricName="areaUnderROC")
auc = evaluator.evaluate(predictions)
print(f"Area under ROC: {auc}")

# Collect predictions and true labels for ROC curve
y_true = predictions.select("target").toPandas()
y_scores = predictions.select("probability").toPandas().apply(lambda x: x[0][1], axis=1)

# Calculate ROC curve
fpr, tpr, _ = roc_curve(y_true, y_scores)
roc_auc = sklearn_auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

```

This **Script 3.5** provides a detailed approach to evaluating the performance of a binary classification model, focusing on the ROC curve and AUC metric. The process involves calculating these metrics using PySpark and visualizing the results to better assess the model's effectiveness.

Key Steps in the Evaluation Process:

1. **Importing Necessary Libraries:**
   The script begins by importing the required libraries, including PySpark's `BinaryClassificationEvaluator` for calculating the AUC score and `sklearn` for additional functions like `roc_curve`. These tools are essential for performing the evaluation and visualization steps.

2. **Calculating AUC with PySpark:**
   The AUC score, a key metric in binary classification, is calculated using PySpark's `BinaryClassificationEvaluator`. The AUC measures the model's ability to differentiate between positive and negative classes, with a score closer to 1 indicating better performance.

3. **Preparing Data for ROC Curve:**
The script extracts the true labels and predicted probabilities from the 'predictions' DataFrame, the output of the model's predictions. This step is crucial as it prepares the data for plotting the ROC curve by organizing the necessary information to calculate false positive and true positive rates.

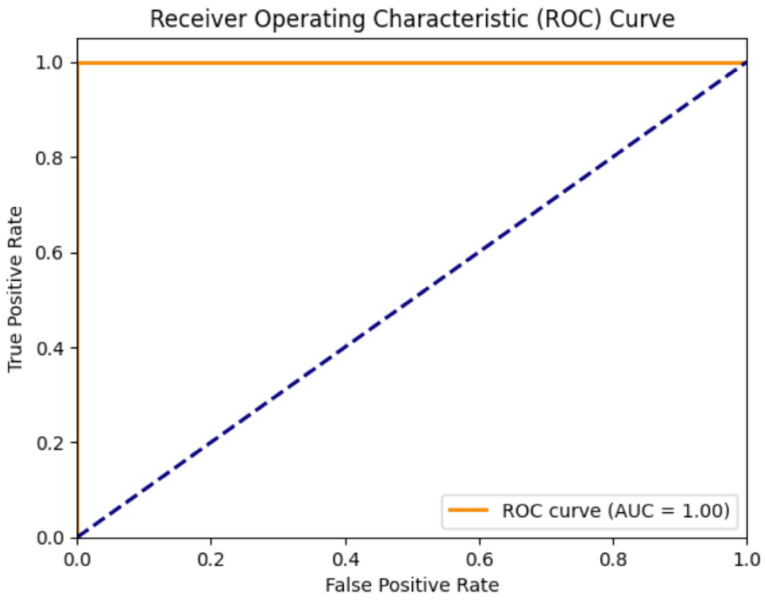
4. **Plotting the ROC Curve:**
Using `sklearn`'s `roc_curve` function, the script calculates the false positive rate (FPR) and true positive rate (TPR) across different classification thresholds. These rates are then used to plot the ROC curve, which visually represents the trade-off between sensitivity (true positive rate) and specificity (1 - false positive rate) at various thresholds.

5. **Visualizing the Results:**
The script uses `matplotlib` to create and display the ROC curve plot. The AUC score is included in the legend, providing a clear indication of the model's overall performance. This visualization helps to assess how well the model can distinguish between the classes, and the curve's shape offers insights into the model's strengths and weaknesses.

Conclusion:
Evaluating a binary classification model using the ROC curve and AUC score provides a comprehensive understanding of its performance. This script calculates these crucial metrics using PySpark and visualizes them, making it easier to interpret the model's ability to discriminate between different classes. The combination of AUC calculation and ROC curve plotting offers a robust framework for assessing model effectiveness, ensuring that the model is accurate and reliable for real-world applications.

Model Evaluation Based on the ROC Curve

Graph/Plot 3.1: ROC Curve for Logistic Regression Model in Weather Forecasting



Graph/Plot 3.1 with the ROC curve graph for the machine learning model shows a perfect classification performance, with an AUC of 1.00. The curve immediately reaches a True Positive Rate of 1.0 at a False Positive Rate of 0, indicating that the model correctly classifies all instances without errors. While this result might seem ideal initially, such flawless performance is sporadic in practical applications and often signals underlying issues that need careful investigation.

Interpreting the Perfect AUC:

1. **Potential Data Leakage:**

Data leakage is the most common cause of a perfect ROC curve and AUC. This occurs when the model has access to information during training that it shouldn't have—information that is too closely related to the target variable. For example, suppose features derived directly from the target variable (shortwave radiation) are included in the model. In that case, the model will memorize the training data instead of learning meaningful patterns, producing artificially high performance.

2. **Overfitting:**

Another potential issue is overfitting, where the model becomes too complex and captures noise or specific details of the training data rather than general patterns. Overfitted models perform exceptionally well on training data but must generalize to new, unseen data, often leading to poor performance when applied in real-world scenarios.

3. **Problem Formulation Issues:**
The simplicity or formulation of the problem contributes to this result. Suppose the problem is too straightforward or the features are too directly related to the target variable. In that case, the model might quickly achieve perfect scores without genuinely understanding the underlying data dynamics. This could mean that the model needs to be challenged more, and the problem might need to be reformulated to make it more complex and realistic.

Steps for Further Investigation:

Given these concerns, it's crucial to conduct a thorough investigation to understand the root cause of this perfect performance:

- **Review Feature Engineering:**
Carefully review the feature engineering process to ensure that no features derived directly or indirectly from the target variable are included in the model. This step is crucial to avoid data leakage and ensure the model learns from appropriate inputs.

- **Rigorous Cross-Validation:**
Implement rigorous cross-validation to test the model on multiple subsets of the data. This process will help identify whether the model's performance is consistent across different data splits and whether it generalizes well beyond the specific training set.

- **Testing on Unseen Data:**
Evaluate the model using entirely new, unseen data. This step is essential to verify that the model performs well outside the data it was trained on and detect any overfitting signs.

- **Reformulate the Problem if Necessary:**
If the problem needs to be more complex, consider reframing it to make it more challenging and reflective of real-world scenarios. This might involve introducing more complexity into the features or changing the prediction task to better capture the nuances of the data.

Conclusion:
While a perfect AUC of 1.00 might initially appear as an ideal outcome, it often points to potential problems such as data leakage, overfitting, or an overly simplistic problem formulation. It's essential to approach such results cautiously and conduct thorough checks to ensure the model's performance is genuine and will generalize well to new data. We can ensure that the model is reliable and effective in real-world applications only through careful validation and testing.

Hyperparameter Optimization for Logistic Regression in Weather Forecasting

Hyperparameter tuning is essential in optimizing machine learning models, particularly in big-data environments where the goal is to achieve the best possible performance by fine-tuning model parameters. In this context, Spark's MLlib provides robust tools to streamline the tuning process, ensuring the model is effective and efficient.

Script 3.6: Shortwave Radiation Prediction: Data Preparation and Model Pipeline

```
print("Step 1: Feature selection and engineering")
# Select relevant features (excluding target-related columns and ID columns)
selected_features = [
    "temperature", "dewpoint", "rain", "snowfall", "surface_pressure",
    "cloudcover_total", "cloudcover_low", "cloudcover_mid", "cloudcover_high",
    "windspeed_10m", "winddirection_10m", "direct_solar_radiation", "diffuse_radiation",
    "latitude", "longitude", "month", "day", "hour"
]
print(f"Selected features: {selected_features}")

# Calculate the mean of shortwave_radiation
mean_shortwave = df_final.agg(avg("shortwave_radiation")).collect()[0][0]
print(f"Mean shortwave radiation: {mean_shortwave}")

# Create binary target variable
df_prepared = df_final.withColumn("target", when(col("shortwave_radiation") > mean_shortwave, 1.0).otherwise(0.0))
print("Binary target variable created")

print("\nStep 2: Splitting data into training, validation, and testing sets")
train_data, val_data, test_data = df_prepared.randomSplit([0.6, 0.2, 0.2], seed=42)
print(f"Train set size: {train_data.count()}")
print(f"Validation set size: {val_data.count()}")
print(f"Test set size: {test_data.count()}")

print("\nStep 3: Creating pipeline for preprocessing and model training")
# Assemble features into a vector
assembler = VectorAssembler(inputCols=selected_features, outputCol="assembled_features")

# Standardize features
scaler = StandardScaler(inputCol="assembled_features", outputCol="scaled_features")

# Define the logistic regression model
lr = LogisticRegression(featuresCol="scaled_features", labelCol="target")

# Create the pipeline
pipeline = Pipeline(stages=[assembler, scaler, lr])
print("Pipeline created")

print("\nStep 4: Hyperparameter tuning with cross-validation")
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.01, 0.1, 1.0]) \
    .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0]) \
    .build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(labelCol="target"),
                          numFolds=5)
print("Cross-validator created")
```

Based on weather data, **Script 3.6** is designed to implement a binary classification pipeline using PySpark's MLlib to predict high or low shortwave radiation levels. The pipeline integrates several critical stages, from data preprocessing to model evaluation, ensuring a comprehensive approach to building a reliable machine-learning model.

Key Components of the Pipeline:

1. **Feature Selection and Data Preparation:**
   The pipeline begins with feature selection and data preparation, ensuring the data is clean and relevant to the modelling task. This step includes handling missing values, normalizing features, and encoding categorical variables, which is crucial for improving model accuracy.

2. **Model Training with Logistic Regression:**
   The pipeline's core is the logistic regression model, chosen for its efficiency and interpretability in binary classification tasks. This model is trained on the prepared dataset to predict whether shortwave radiation levels will be high or low.

3. **Hyperparameter Tuning via Cross-Validation:**
   To optimize the logistic regression model, the script employs PySpark's cross-validation capabilities, using `CrossValidator` and `ParamGridBuilder` to explore different combinations of hyperparameters. The tuning process tests various configurations to identify the parameters that yield the highest AUC (Area Under the Curve) score, ensuring the model's robustness and accuracy.

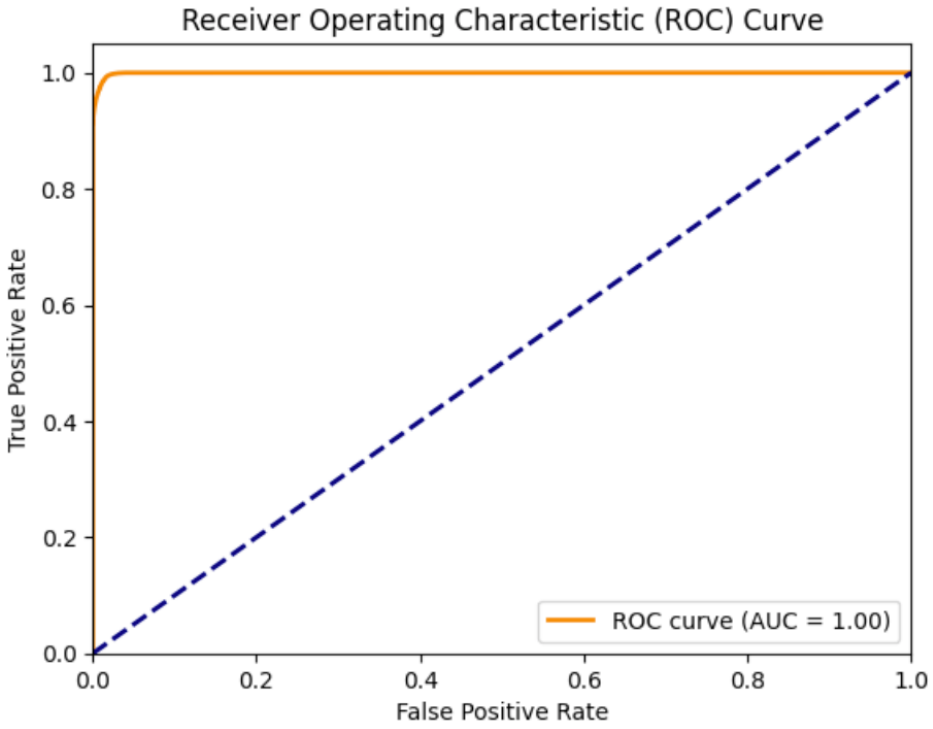
4. **Comprehensive Model Evaluation:**
   After tuning, the model's performance is evaluated on validation and test sets, with AUC as the primary metric. The script also includes additional evaluation methods, such as plotting the ROC curve visually representing the model's classification performance.

5. **Additional Analyses:**
   Beyond standard evaluation, the script delves into deeper analyses, such as examining feature importance to understand which variables most influence the predictions, checking class balance to ensure that the model is not biased towards one class, and visualizing feature distributions to gain insights into the data's structure.

Conclusion:
This pipeline offers a thorough approach to building and understanding a machine-learning model for weather-related predictions by integrating data preprocessing, model development, hyperparameter tuning, and in-depth analysis. PySpark's MLlib is leveraged to handle the complexities of big data, making this pipeline both scalable and effective in a real-world, big-data environment. The resulting model is well-optimized for predicting shortwave radiation levels through careful tuning and comprehensive evaluation, providing valuable insights for weather forecasting and energy management.

Model Evaluation Based on the ROC Curve

Graph/Plot 3.2: ROC Curve for Logistic Regression Model in Weather Forecasting



The machine learning model developed to predict shortwave radiation demonstrates impressive accuracy and highlights several critical insights for understanding the model's performance and considering its practical application. Below are the key findings:

1. **Model Performance:**
   - The model exhibits outstanding performance, with AUC scores of **0.9994** (**Graph/Plot 3.2)** on both the validation and test sets. These high AUC scores indicate the model's strong predictive power and ability to distinguish between the classes accurately.

2. **Feature Importance:**
   - The most significant predictors for shortwave radiation are:
     - **Diffuse Radiation:** Importance score of 3.35
     - **Direct Solar Radiation:** Importance score of 2.60
     - **Total Cloud Cover:** Importance score of -0.36
   - Other features were found to have zero importance, suggesting that they do not contribute significantly to the model's predictions.

3. **Model Parameters:**
   - The best-performing model uses a regularization parameter of **0.01** and an elastic net parameter of **1.0**, indicating that a **Lasso regression** approach was employed. This choice of parameters suggests a preference for simplicity and sparsity in the model, penalizing the complexity of the features.

4. **Class Balance:**
- The training dataset is imbalanced, with **739,317 samples** in class 0.0 and **288,219** in class 1.0. This imbalance could affect the model's performance, particularly in predicting the minority class.

5. **Potential Data Leakage:**
- There are strong correlations between the target variable (shortwave radiation) and some of the predictors:
  - **Direct Solar Radiation:** Correlation of 0.97
  - **Diffuse Radiation:** Correlation of 0.83
- These strong correlations suggest potential data leakage or that these predictors are direct components of shortwave radiation, which could lead to overfitting and limit the model's practical applicability.

6. **Other Correlations:**
- **Temperature** shows a moderate positive correlation (0.45) with shortwave radiation.
- **Dewpoint** also has a moderate positive correlation (0.25) with shortwave radiation.
- **Cloud Cover** shows a negative correlation, indicating that as cloud cover increases, shortwave radiation typically decreases.

7. **Temporal Factors:**
- The **hour of the day** has a weak positive correlation (0.11) with shortwave radiation, aligning with expectations given the daily solar cycle's influence on radiation levels.

8. **Feature Selection:**
- The model utilizes **18 features**, which include various weather parameters and temporal factors such as the month, day, and hour. These features were selected for their relevance in predicting shortwave radiation levels.

9. **Data Split:**
- The dataset was split into training **(60%)**, **validation (20%)**, and **test (20%)** sets. This split ensures that the model is evaluated on unseen data, providing a robust measure of its generalization capability.

10. **Binary Classification:**
- The prediction task was framed as a binary classification problem, likely focusing on predicting whether shortwave radiation levels would be high or low. This approach simplifies the problem but necessitates carefully considering how thresholds are set and interpreted.

Conclusion:
The machine learning model for predicting shortwave radiation demonstrates exceptional accuracy, with near-perfect AUC scores on both validation and test datasets. The model's most critical predictors include diffuse, direct solar radiation and total cloud cover. However, the strong correlations between these predictors and the target variable raise concerns about

potential data leakage, which could lead to overfitting or reduce the model's real-world applicability.

Moreover, the model's reliance on these closely related features suggests that it may capture direct components of shortwave radiation rather than more complex, indirect relationships. As a result, while the model's performance metrics are impressive, its practical application should be approached with caution, particularly in contexts where overfitting or feature redundancy could undermine its utility. Careful interpretation and further validation are necessary to ensure the model provides meaningful and actionable insights in real-world scenarios.

4.6 LSTM (Long Short-Term Memory) in Energy Forecasting

LSTM (Long Short-Term Memory) networks are precious for energy forecasting projects due to their ability to handle complex time series data. These neural networks excel at capturing long-term dependencies in sequential data, making them ideal for energy consumption or production prediction where patterns may depend on events from hours, days, or months ago. Learning from recent and older data points, LSTMs overcome the vanishing gradient problem. They adaptively learn to forget irrelevant information and remember important details, which is crucial in energy systems where both short-term fluctuations and long-term trends matter. LSTMs can process multiple input variables simultaneously, capture non-linear relationships, and effectively learn seasonal and cyclical patterns common in energy data. They are robust to noise, scalable to handle large amounts of historical data, and suitable for single- and multi-step forecasting. LSTMs can also be integrated with other neural network types for hybrid models. While they require careful data preprocessing, architecture design, and hyperparameter tuning, LSTMs offer the potential for more accurate and robust energy forecasting models compared to more straightforward approaches, particularly in scenarios with complex temporal dependencies.

Script 3.7: LSTM Time Series Forecasting with PySpark and Keras

```
# Initialize Spark session
spark = SparkSession.builder.appName("LSTM with PySpark - 500k records").getOrCreate()

def print_df_info(df, name):
    print(f"\n--- {name} Info ---")
    print(f"Count: {df.count()}")
    print("Schema:")
    df.printSchema()
    print("Sample data:")
    df.show(5, truncate=False)

try:
    # Step 1: Load and prepare data
    print("\nStep 1: Loading and preparing data...")
    df = spark.read.csv("/content/drive/MyDrive/big_data_project/historical_weather.csv", header=True, inferSchema=True)

    # Randomly sample 500,000 records
    df = df.orderBy(rand()).limit(500000)

    pandas_df = df.select(col('shortwave_radiation')).toPandas()
    data = pandas_df.values

    # Step 2: Normalize the data
    print("\nStep 2: Normalizing data...")
    scaler = MinMaxScaler(feature_range=(0, 1))
    scaled_data = scaler.fit_transform(data)

    # Step 3: Create sequences
    print("\nStep 3: Creating sequences...")
    sequence_length = 2  # Adjust as needed
    generator = TimeseriesGenerator(scaled_data, scaled_data, length=sequence_length, batch_size=32)

    # Step 4: Build LSTM model
    print("\nStep 4: Building LSTM model...")
    model = Sequential()
    model.add(LSTM(50, activation='relu', input_shape=(sequence_length, 1)))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mse')

    # Step 5: Train the model
    print("\nStep 5: Training the model...")
    model.fit(generator, epochs=5)

    # Step 6: Make predictions
    print("\nStep 6: Making predictions...")
    predictions = model.predict(generator)
    print(f"Predictions shape: {predictions.shape}")
    print(f"Predictions sample: {predictions[:5]}")
```

This Python **Script 3.7** implements an LSTM (Long Short-Term Memory) model using PySpark and TensorFlow for time series prediction on weather data. It begins by initializing a Spark session and loading 500,000 randomly sampled records from a CSV file. The data is then normalized, and time series sequences are created for LSTM input. The code builds and trains an LSTM model, makes predictions, and converts the results to the original scale. It then processes the results using PySpark, joining the predictions with the original data and evaluating the model's performance using metrics such as RMSE, MAE, R2, and MAPE. The script includes error handling and ensures proper cleanup of the Spark session. Overall, it demonstrates an end-to-end workflow for large-scale time series prediction, combining the big data processing capabilities of PySpark with the deep learning power of TensorFlow.

Evaluating the Performance of the LSTM (Long Short-Term Memory) Model

LSTM Weather Prediction Model Analysis
Our LSTM model for weather prediction has shown significant underperformance, failing to learn meaningful patterns from the dataset. This report details our findings and provides recommendations for improvement.

1. **Data Processing**
- Dataset: Successfully processed 500,000 records from the weather dataset.
- Preprocessing: Data and sequences created for LSTM input were normalised.

2. **Model Architecture and Training**
- Model: Long Short-Term Memory (LSTM) neural network
- Training Duration: 5 epochs
- Training Performance:
  - Initial loss: 0.0458 (Epoch 1)
  - Final loss: 0.0454 (Epoch 5)
  - Observation: Minimal improvement during training, suggesting potential issues with model learning or data complexity.

3. **Prediction Analysis**
- Number of Predictions: 499,998
- Prediction Behavior: All predictions converged to a constant value (104.3859634399414)
- Interpretation: The model failed to capture temporal patterns or variability in the data

4. **Data Integration**
- Joined Dataset Size: 249,856 records (from original 500,000)
- Data Loss: Approximately 50% of records were lost during the join process
- Potential Causes: Indexing misalignment or timestamp inconsistencies

5. **Data Quality Assessment**
- Null/NaN Values: None detected in the result DataFrame
- Positive Aspect: Maintains data integrity post-processing

6. **Model Evaluation Metrics**

Metric	Value	Interpretation
RMSE	180.29	High error magnitude
MAE	134.98	Significant average deviation
R^2	-0.0002	Model performs worse than a simple mean
MAPE	549.77%	Extreme percentage errors

7. **In-depth Prediction Analysis**
- Constant Prediction: 104.3859634399414 for all data points
- MAPE Variability: Wide range, with extremely high values for some predictions
- Implication: The model is insensitive to input features and temporal patterns

Conclusion
The LSTM model must capture meaningful patterns in the weather data, resulting in a constant output for all predictions. A constant output indicates fundamental issues with the model architecture, training process, or data preparation.

Recommendations:

1. **Data Preprocessing Review**:
   - Validate normalisation techniques
   - Ensure proper sequence creation for LSTM input
   - Investigate and rectify data loss during the joining process

2. **Model Architecture Optimization**:
   - Experiment with different LSTM configurations (layers, units)
   - Consider alternative architectures (e.g., GRU, Transformer)
   - Implement regularisation techniques to prevent overfitting

3. **Training Process Enhancement**:
   - Increase the number of epochs
   - Implement learning rate scheduling
   - Use early stopping to prevent overfitting

4. **Feature Engineering**:
   - Conduct feature importance analysis
   - Create domain-specific features relevant to weather prediction
   - Consider dimensionality reduction techniques

5. **Cross-Validation**:
   - Implement k-fold cross-validation to ensure model generalisation

6. **Data Quality and Quantity**:
   - Investigate potential data quality issues
   - Consider acquiring additional data if available

By systematically addressing these recommendations, we aim to develop a more accurate and reliable weather prediction model using LSTM or alternative approaches.

Script 3.8: LSTM Hyperparameter Tuning with Random Search

```python
# Step 4: Define hyperparameter search
print("\nStep 4: Defining hyperparameter search...")
param_grid = {
    'lstm_units': [50, 100],
    'dropout_rate': [0.1, 0.2],
    'learning_rate': [0.001, 0.01],
    'batch_size': [64, 128],
    'epochs': [30]  # Fixed number of epochs, we'll use early stopping
}

# Step 5: Perform random search
print("\nStep 5: Performing random search...")
n_iter = 5  # Number of random combinations to try
best_score = float('inf')
best_params = {}

for i in range(n_iter):
    current_params = {k: random.choice(v) for k, v in param_grid.items()}
    print(f"Testing parameters: {current_params}")

    model = create_model(current_params['lstm_units'], current_params['dropout_rate'], current_params['learning_rate'])

    early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

    history = model.fit(X_train, y_train, epochs=current_params['epochs'],
                        batch_size=current_params['batch_size'], validation_split=0.2,
                        callbacks=[early_stopping], verbose=0)

    val_loss = min(history.history['val_loss'])
    if val_loss < best_score:
        best_score = val_loss
        best_params = current_params

print("Best parameters found: ", best_params)
print("Best validation loss: ", best_score)

# Step 6: Train the best model
print("\nStep 6: Training the best model...")
best_model = create_model(best_params['lstm_units'], best_params['dropout_rate'], best_params['learning_rate'])
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
best_model.fit(X_train, y_train, epochs=100,  # Increased max epochs
               batch_size=best_params['batch_size'], validation_split=0.2,
               callbacks=[early_stopping], verbose=1)

# Step 7: Make predictions
print("\nStep 7: Making predictions...")
predictions = best_model.predict(X_test)
print(f"Predictions shape: {predictions.shape}")
print(f"Predictions sample: {predictions[:5]}")
```

Script 3.8 implements a time series forecasting pipeline using PySpark and TensorFlow, focusing on predicting shortwave radiation from historical weather data. It loads and preprocesses a large dataset, creates sequences for LSTM input, and performs a random search to optimize hyperparameters. The best LSTM model is then trained with early stopping, used to make predictions on a test set, and evaluated using various metrics. The pipeline leverages Spark for efficient data handling and TensorFlow for model training, combining big data processing with deep learning techniques. The script includes error handling, informative logging, and proper resource management, making it suitable for processing large-scale time series data in a distributed environment.

Analysis of LSTM Weather Prediction Model Results

1. **Hyperparameter Tuning:**
   The best parameters were 50 LSTM units, 0.1 dropout rate, 0.001 learning rate, batch size 128, and 30 epochs.
   - The best validation loss achieved during tuning was 0.048551276326179504.

2. **Model Training:**
   - The model training stopped after 16 epochs, likely due to early stopping.
   - The training and validation losses were close (around 0.0486), indicating the model was not overfitting.

3. **Predictions:**
   - The model's predictions have very low variance, with values mostly around 105-106.
   - This suggests the model is underfitting, predicting almost a constant value regardless of input.

4. **Performance Metrics:**
   - RMSE: 180.86197585111898
   - MAE: 136.17663898980166
   - R2: 3.1632417388793854e-06
   - MAPE: 566.9630281106342%

   These metrics indicate extremely poor model performance:
   - High RMSE and MAE suggest significant prediction errors.
   - R2 score very close to 0 (slightly positive) indicates the model performs only marginally better than a simple mean predictor.
   - Extremely high MAPE suggests substantial percentage errors in predictions.

5. **Specific Issues:**
   - The model consistently predicts around 105-106 values, regardless of the actual value.
   - It fails to predict zero values, which appear frequently in the data.
   - The model does not capture the wide range of the target variable (from 0 to 592 in the sample shown).

6. **Data Characteristics:**
   - The target variable (likely 'shortwave_radiation') has many zero values.
   - The target variable has a wide range (0 to 592 in the sample), which the model fails to capture.

7. **Error Analysis:**
   - MAPE is undefined (NULL) for actual zero values, which are frequent in the dataset.
   - The MAPE is often high for non-zero actual values, sometimes exceeding 5000%.

- The model's constant-like predictions result in wildly varying error percentages depending on the actual value.

Conclusions:
1. The LSTM model needs to learn meaningful patterns from the data.
2. It behaves almost like a constant predictor, suggesting fundamental issues with the model architecture or data preprocessing.
3. The many zero values in the target variable might be causing issues with the model's learning process.
4. The model does not capture the wide range of the target variable (0 to 592+).

Recommendations:
1. Revisit the data preprocessing steps, mainly how zero values and the target variable's wide range are handled.
2. Consider feature engineering or including additional relevant features if available.
3. Experiment with different model architectures or even try other types of models (e.g., Random Forests, Gradient Boosting) that might better handle zero values and a wide range.
4. Investigate the distribution of the target variable and consider techniques like log transformation if it is highly skewed.
5. Analyse the temporal aspects of the data more closely, as the LSTM might need to capture the time-based patterns more effectively.
6. Consider using a different evaluation metric more suitable for data with many zero values.
7. Review the data normalization process, as the predictions are being converted to a scale that does not match the original data range.

4.7 CNN (Convolutional Neural Network) in Energy Forecasting

Convolutional Neural Networks (CNNs) can be valuable for energy forecasting projects despite being more commonly associated with image processing tasks. CNNs excel at pattern recognition and feature extraction, making them helpful in identifying recurring patterns in energy consumption or production data. They can effectively process multiple input variables simultaneously, which is common in energy forecasting. CNNs capture local patterns in time series data, which is crucial for short-term energy forecasting. They often have fewer parameters than fully connected networks, reducing the risk of overfitting. They demonstrate temporal invariance, learning patterns regardless of where they occur in the sequence. CNNs can be adapted for multi-step forecasting and handle non-linear relationships in data. They are scalable, efficiently processing large datasets of historical energy data. CNNs can also be combined with other models like LSTMs to create robust hybrid architectures. Implementing CNNs for energy forecasting requires careful data preparation, model architecture design, and hyperparameter tuning, and they offer the potential to capture complex patterns that simpler models might miss. However, their use should be justified by improved forecasting accuracy compared to less complex models.

Script 3.9: CNN Model Construction and Prediction Pipeline

```
    # Step 4: Build CNN model
    print("\nStep 4: Building CNN model...")
    model = Sequential([
        Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(sequence_length, 1)),
        MaxPooling1D(pool_size=2),
        Conv1D(filters=32, kernel_size=3, activation='relu'),
        MaxPooling1D(pool_size=2),
        Flatten(),
        Dense(50, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer='adam', loss='mse')

    # Step 5: Train the model
    print("\nStep 5: Training the model...")
    model.fit(generator, epochs=10)

    # Step 6: Make predictions
    print("\nStep 6: Making predictions...")
    predictions = model.predict(generator)
    print(f"Predictions shape: {predictions.shape}")
    print(f"Predictions sample: {predictions[:5]}")

    # Step 7: Convert predictions back to original scale
    print("\nStep 7: Converting predictions to original scale...")
    predictions = scaler.inverse_transform(predictions)
    print(f"Inverse transformed predictions shape: {predictions.shape}")
    print(f"Inverse transformed predictions sample: {predictions[:5]}")

    # Step 8: Convert predictions to Spark DataFrame
    print("\nStep 8: Converting predictions to Spark DataFrame...")
    predictions_df = spark.createDataFrame(predictions.tolist(), ["prediction"])
    print_df_info(predictions_df, "Predictions DataFrame")

    # Step 9: Join with original data
    print("\nStep 9: Joining predictions with original data...")
    print_df_info(df, "Original DataFrame")

    result_df = df.withColumn("row_id", monotonically_increasing_id()).join(
        predictions_df.withColumn("row_id", monotonically_increasing_id()),
        "row_id"
    ).drop("row_id")
    print_df_info(result_df, "Result DataFrame")
```

Script 3.9 implements a Convolutional Neural Network (CNN) using PySpark and TensorFlow to predict shortwave radiation based on historical weather data. It loads and prepares the data using PySpark, normalizes it, creates time series sequences, builds and trains a CNN model, makes predictions, and then post-processes the results. The script joins the predictions with the original data, performs data quality checks, and evaluates the model's performance using various regression metrics. Throughout the process, it leverages PySpark's distributed computing capabilities for data handling and TensorFlow for deep learning. It combines big data processing with advanced machine learning techniques to create a scalable solution for time series prediction on weather data. The script includes error handling and informative print statements to track progress and catch issues, culminating in a display of the final results that include both the original data and the model's predictions.

CNN Model Development and Evaluation for Weather Data Prediction

1. **Data Processing**: The process involved loading, preparing, and normalizing data, followed by creating sequences for a CNN (Convolutional Neural Network) model.

2. **Model Training**:
   - A CNN model was built and trained for ten epochs.
   - The training process showed a decreasing loss over the epochs, starting from 0.0039 in the first epoch and ending at 0.0028 in the tenth epoch.
   - Each epoch processed 53,462 steps, indicating a large dataset.

3. **Predictions**:
   - Predictions were made on 1,710,778 data points.
   - The predictions were then converted back to the original scale.

4. **Data Integration**:
   - The predictions were converted to a Spark DataFrame and joined with the original data.
   - The original data included weather-related features such as temperature, dewpoint, rain, snowfall, surface pressure, cloud cover, wind speed and direction, solar radiation, and geographical coordinates.

5. **Data Quality**:
   - A check for NaN or null values was performed, and no null or NaN values were found in the final dataset.

6. **Model Evaluation**:
   - The model's performance was evaluated using several metrics:
     - RMSE (Root Mean Square Error): 163.76%
     - MAE (Mean Absolute Error): 82.89%
     - R2 (Coefficient of Determination): 0.1563%
     - MAPE (Mean Absolute Percentage Error): 349.52%

7. **Results Interpretation**:
   - The high MAPE (349.52%) suggests that the model's predictions have a large average percentage error compared to the actual values.
   - The low R2 score (0.1563) indicates that the model explains only about 15.63% of the variability in the target variable, suggesting poor fit.
   - The RMSE and MAE values are pretty high, but knowing the scale of the target variable is necessary to interpret their significance.

8. **Final Dataset**:
   - The final dataset includes the original features and the model's predictions.

- Some rows in the sample data show NULL values for MAPE, which might indicate instances where the calculation was impossible (e.g., division by zero).

In conclusion, while the model training process was completed successfully, the evaluation metrics suggest the model's poor predictive performance. The high error rates and low R2 score indicate that the model is not effectively capturing the underlying patterns in the data. Further investigation into the data, feature engineering, or trying different model architectures might be necessary to improve the predictions.

CNN-Based Shortwave Radiation Prediction with Hyperparameter Tuning

Script 3.10: CNN Hyperparameter Tuning with Grid Search

```
def create_model(conv1_filters, conv1_kernel, dense_units):
    model = keras.Sequential([
        layers.Input(shape=(24, 1)),
        layers.Conv1D(filters=conv1_filters, kernel_size=conv1_kernel, activation='relu'),
        layers.GlobalMaxPooling1D(),
        layers.Dense(dense_units, activation='relu'),
        layers.Dense(1, activation='relu')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

def train_and_evaluate(model, X_train, y_train, X_val, y_val, epochs=50, batch_size=128):
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
    history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
                        validation_data=(X_val, y_val), callbacks=[early_stop], verbose=0)
    val_loss = history.history['val_loss'][-1]
    return val_loss

def main():
    start_time = time.time()
    print("\nStep 5: Starting main function")
    X_train, X_val, X_test, y_train, y_val, y_test, scaler = load_and_prepare_data()

    print("\nStep 6: Defining hyperparameter grid")
    conv1_filters_options = [16, 32, 64]
    conv1_kernel_options = [3, 5, 7]
    dense_units_options = [32, 64, 128]

    param_grid = [
        {
            'conv1_filters': random.choice(conv1_filters_options),
            'conv1_kernel': random.choice(conv1_kernel_options),
            'dense_units': random.choice(dense_units_options)
        }
        for _ in range(10)
    ]

    print("\nStep 7: Starting grid search")
    best_loss = float('inf')
    best_params = None

    total_combinations = len(param_grid)
    print(f"Total number of hyperparameter combinations: {total_combinations}")
```

Script 3.10 implements a Convolutional Neural Network (CNN) for time series prediction, specifically for predicting shortwave radiation based on historical weather data.

1. **GPU Check**: It first checks for GPU availability to optimize computations.

2. **Data Loading and Preparation**:
   - Loads historical weather data from a CSV file.

- Extracts the 'shortwave_radiation' column.
- Normalizes the data using MinMaxScaler.
- Prepares sequences for CNN input with a sequence length of 24.
- Splits the data into training, validation, and test sets.

3. **Model Creation**:
- Defines a function to create a CNN model with configurable hyperparameters.
- The model architecture includes a 1D convolutional layer, global max pooling, and dense layers.

4. **Training and Evaluation**:
- Implements a function to train the model with early stopping to prevent overfitting.

5. **Hyperparameter Tuning**:
- Performs a randomized grid search over a set of hyperparameters:
  - Convolutional layer filters: [16, 32, 64]
  - Convolutional kernel size: [3, 5, 7]
  - Dense layer units: [32, 64, 128]
- Tries ten random combinations of these hyperparameters.

6. **Best Model Selection**:
- Selects the best model based on validation loss.

7. **Final Training and Evaluation**:
- Trains the best model on the combined training and validation data.
- Evaluates the final model on the test set.

8. **Predictions**:
- Makes predictions using the best model on the test set.
- Inverses the scaling to get predictions in the original scale.

9. **Performance Tracking**:
- Measures and reports the total execution time of the script.

This script essentially automates the process of building, tuning, and evaluating a CNN model for time series prediction. It focuses on optimizing the model's performance through hyperparameter tuning.

CNN Model Optimization Report: Hyperparameter Tuning and Performance Analysis

1. **Hyperparameter Tuning**:
- A grid search was performed with ten different hyperparameter combinations.
- The best hyperparameters found were:
  - conv1_filters: 64
  - conv1_kernel: 5

- dense_units: 32

2. **Model Performance**:
   - The grid search process showed continuous improvement in the model's performance:
     - Initial best loss: 0.060453880578279495
     - Final best validation loss: 0.002648838795721531
   - This represents a significant improvement in the model's performance during the tuning process.

3. **Test Set Evaluation**:
   - The best model's performance on the test set resulted in a loss of 0.06060829758644104.
   - This is higher than the validation loss, which might indicate some overfitting.

4. **Predictions**:
   - The model made predictions on 10,693 samples.
   - A sample of predictions was provided, showing outputs of [0.] for five instances.
   - These outputs suggest the model predicts binary outcomes (0 or 1), with 0 representing one class.

5. **Execution Time**:
   - The total execution time was 14,408.38 seconds (about 4 hours).
   - This long execution time is likely due to the CPU instead of the GPU for computations.

6. **Model Architecture**:
   - The model is a convolutional neural network (CNN) with one convolutional layer and one dense layer.

The results suggest that the hyperparameter tuning process improved the model's performance on the validation set. However, the higher loss on the test set indicates an opportunity for further enhancement in the model's generalization. This insight should inspire us to continue refining our model. The prolonged execution time suggests that future runs could benefit significantly from GPU acceleration.

4.8 MLPs (Multilayer Perceptron's) in Energy Forecasting

Multilayer Perceptron's (MLPs), also known as feedforward neural networks, are valuable for energy forecasting projects due to their versatility and ability to handle complex data relationships. MLPs excel at capturing non-linear interactions in energy systems where multiple factors interplay intricately. They offer flexibility in handling various input variables and can be used for regression and classification tasks in energy forecasting. MLPs automatically learn relevant features from input data, potentially uncovering hidden patterns. They are scalable and capable of processing large datasets and high-dimensional input spaces, making them suitable for complex energy forecasting scenarios. MLPs adapt to evolving consumption patterns and can be configured for multi-output prediction, allowing simultaneous forecasting of multiple energy variables. With proper preprocessing, they can handle datasets with missing values shared in real-world energy data. MLPs can be effectively combined with other models in ensemble methods to improve overall forecasting accuracy. They balance model complexity and interpretability compared to more advanced deep learning architectures. The architecture of MLPs is easily customizable to suit specific energy forecasting tasks. While implementing MLPs requires careful consideration of data preprocessing, architecture design, and hyperparameter tuning, they provide a powerful tool for capturing complex patterns in energy data, bridging the gap between simple statistical models and more advanced deep learning approaches.

Script 3.11: Time Series Prediction Using MLP and Spark DataFrame Processing

```
    # Step 4: Build MLP model
    print("\nStep 4: Building MLP model...")
    model = Sequential([
        Dense(64, activation='relu', input_shape=(sequence_length, 1)),
        Dropout(0.2),
        Dense(32, activation='relu'),
        Dropout(0.2),
        Dense(16, activation='relu'),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

    # Step 5: Train the model
    print("\nStep 5: Training the model...")
    model.fit(X, y, epochs=10, batch_size=32, validation_split=0.2, verbose=1)

    # Step 6: Make predictions
    print("\nStep 6: Making predictions...")
    predictions = model.predict(X)
    print(f"Predictions shape: {predictions.shape}")
    print(f"Predictions sample: {predictions[:5]}")

    # Step 7: Convert predictions back to original scale
    print("\nStep 7: Converting predictions to original scale...")
    # Reshape predictions to 2D array
    predictions_2d = predictions.reshape(-1, 1)
    predictions_original = scaler.inverse_transform(predictions_2d)
    print(f"Inverse transformed predictions shape: {predictions_original.shape}")
    print(f"Inverse transformed predictions sample: {predictions_original[:5]}")

    # Step 8: Convert predictions to Spark DataFrame
    print("\nStep 8: Converting predictions to Spark DataFrame...")
    predictions_df = spark.createDataFrame(predictions_original.tolist(), ["prediction"])
    print_df_info(predictions_df, "Predictions DataFrame")

    # Step 9: Join with original data
    print("\nStep 9: Joining predictions with original data...")
    print_df_info(df, "Original DataFrame")

    # Adjust the number of rows in predictions_df to match the original df
    predictions_df = predictions_df.limit(df.count() - sequence_length)

    result_df = df.withColumn("row_id", monotonically_increasing_id()).join(
        predictions_df.withColumn("row_id", monotonically_increasing_id()),
        "row_id"
    ).drop("row_id")
    print_df_info(result_df, "Result DataFrame")
```

Script 3.11 implements a Multilayer Perceptron (MLP) neural network using PySpark and TensorFlow to predict shortwave radiation based on historical weather data. It loads and preprocesses the data using PySpark, normalizes it, creates time series sequences, builds and trains an MLP model, generates predictions, and then post-processes the results. The script joins the predictions with the original data, performs data quality checks, and evaluates the model's performance using various regression metrics (RMSE, MAE, R2, MAPE). It leverages PySpark's distributed computing capabilities for data handling and TensorFlow for deep learning. It combines big data processing with neural network techniques to create a scalable solution for time series prediction on weather data. The script includes error handling and

informative print statements to track progress and catch issues, concluding with a display of the final results, including the original data and the model's predictions.

MLPs Model Development and Evaluation for Weather Data Prediction

1. **Data Processing and Model Training**:
   - The process involved loading and preparing data, normalizing it, and preparing input/output sequences.
   - An MLP (Multi-Layer Perceptron) model was built and trained for ten epochs.
   - The model showed relatively stable loss values during training, with the final training loss at 0.0080 and validation loss at 0.0149.

2. **Predictions**:
   - The model made predictions on a large dataset (1,710,792 samples).
   - The predictions were then converted to the original scale and transformed into a Spark DataFrame.

3. **Data Integration**:
   - The predictions were joined with the original data, resulting in a DataFrame with 877,226 rows.
   - The final DataFrame includes weather-related features such as temperature, dewpoint, rain, snowfall, surface pressure, cloud cover, wind speed, solar radiation, and geographical coordinates.

4. **Model Evaluation**:
   - The model's performance was evaluated using several metrics:
     - RMSE (Root Mean Square Error): 215.10200314062394
     - MAE (Mean Absolute Error): 142.15454759745697
     - R2 (Coefficient of Determination): -0.33370737838875697
     - MAPE (Mean Absolute Percentage Error): 436.70528260150684%

5. **Interpretation of Results**:
   - The high RMSE and MAE values suggest that the model's predictions have significant errors.
   The negative R2 value indicates that the model performs worse than a horizontal line (the mean of the data). This suggests that the model is not effectively capturing the underlying patterns in the data.
   - The highly high MAPE (436.71%) further confirms the poor predictive performance of the model.

6. **Data Quality**:
   - A check for NaN or null values in the final DataFrame showed no missing values across all columns.

7. **Sample Predictions**:
   - The sample predictions shown in the final results table demonstrate some inconsistencies:
   - Many predictions have the same value (6.668169021606445), which could indicate a problem with the model's ability to differentiate between different input conditions.
   - Some predictions show significantly different values (e.g., 239.24911499023438), but with context, it is easier to determine if these are more accurate or just outliers.

In conclusion, the resulting model's performance could improve, but the data processing and model training steps were executed without apparent issues. The high error rates and negative R2 value suggest that the model must effectively capture the data relationships. Further investigation into the data, feature selection, model architecture, and training process would be necessary to improve the results.

MLPs-Based Shortwave Radiation Prediction with Hyperparameter Tuning

Script 3.12: Time Series Prediction using Random Forest and LSTM Neural Network

```
        # Step 2: Train Random Forest model
        print("\nStep 2: Training Random Forest model...")
        rf = RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42, n_jobs=-1)
        rf.fit(X_train, y_train)

        # Make predictions
        rf_predictions = rf.predict(X_test)

        # Evaluate Random Forest model
        rf_rmse = np.sqrt(mean_squared_error(y_test, rf_predictions))
        print(f"Random Forest RMSE: {rf_rmse}")

        # Step 3: Prepare data for neural network
        print("\nStep 3: Preparing data for neural network...")
        scaler = MinMaxScaler(feature_range=(0, 1))
        X_scaled = scaler.fit_transform(X)
        y_scaled = scaler.fit_transform(y.values.reshape(-1, 1))

        # Prepare input/output sequences
        sequence_length = 24
        X_seq, y_seq = [], []
        for i in range(len(X_scaled) - sequence_length):
            X_seq.append(X_scaled[i:i+sequence_length])
            y_seq.append(y_scaled[i+sequence_length])
        X_seq, y_seq = np.array(X_seq), np.array(y_seq)

        # Split the data
        X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42)

        # Step 4: Build neural network model
        print("\nStep 4: Building neural network model...")
        model = Sequential([
            LSTM(64, activation='relu', input_shape=(sequence_length, len(feature_cols))),
            Dense(32, activation='relu'),
            Dense(1)
        ])
        model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

        # Step 5: Train the model with early stopping
        print("\nStep 5: Training the model...")
        early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
        history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, callbacks=[early_stopping], verbose=1)

        # Step 6: Make predictions
        print("\nStep 6: Making predictions...")
        predictions = model.predict(X_test)
```

Script 3.12 uses historical weather data to implement a machine-learning pipeline for predicting shortwave radiation.

1. **Data Loading and Preparation**:
   - Loads a CSV file containing historical weather data.
   - Performs feature engineering, creating new features like 'prev_shortwave_radiation', 'radiation_change', 'day_of_year', and 'hour'.
   - Handles null values by filling them with zeros.

2. **Random Forest Model**:
   - Splits the data into training and testing sets.
   - Trains a Random Forest Regressor model.
   - Makes predictions and evaluates the model using Root Mean Square Error (RMSE).

3. **Neural Network Preparation**:
   - Scales the features and target variable using MinMaxScaler.
   - Creates data sequences for time series prediction (sequence length 24).

4. **Neural Network Model**:
   - Builds a sequential model using LSTM (Long Short-Term Memory) and Dense layers.
   - Compiles the model with Adam optimizer and Mean Squared Error loss.

5. **Neural Network Training**:
   - Trains the model using the prepared sequences.
   - Implements early stopping to prevent overfitting.

6. **Prediction and Evaluation**:
   - Makes predictions using the trained neural network.
   - Converts predictions back to the original scale.
   - Evaluate the model using RMSE, MAE (Mean Absolute Error), and R2 score.

7. **Feature Importance**:
   - Extracts and prints feature importance from the Random Forest model.

The script includes error handling and informative print statements to track the progress of each step. It is designed to work with a large dataset but limits the initial data load to 100,000 rows for efficiency.

This pipeline combines traditional machine learning (Random Forest) with deep learning (LSTM neural network) approaches to predict shortwave radiation based on weather-related features. Using both methods allows for comparison and potentially improved predictions by leveraging the strengths of each approach.

MLPs Model Optimization Report: Hyperparameter Tuning and Performance Analysis

1. **Data Preparation**:
   - The original dataset contained 100,000 rows and 18 columns.
   - The data was processed and reduced to 14 relevant features for the analysis.

2. **Random Forest Model**:
   - A Random Forest model was initially trained, resulting in an RMSE of 2.7366556169460563.

3. **Neural Network Model**:
   - A neural network model was built and trained for 50 epochs.
   - The model showed continuous improvement in training and validation loss throughout the training process.

4. **Model Performance**:
   - The final model evaluation metrics on the test set were:
     - RMSE (Root Mean Square Error): 14.972185649290072
     - MAE (Mean Absolute Error): 6.858076266575379
     - R2 Score: 0.9887387602537765
   - The high R2 score (0.9887) indicates that the model explains 98.87% of the variance in the target variable, suggesting a perfect fit.

5. **Feature Importance**:
   - The Random Forest model was used to determine feature importance:
     - The most important feature by far was 'prev_shortwave_radiation', which had an importance of 0.7938 (79.38%).
     - The second most important feature was 'radiation_change', which had an importance of 0.2032 (20.32%).
     - Other features had much lower importance, with 'hour' being the next most important at 0.0024 (0.24%).
     - Some features like 'snowfall' showed zero importance in the model.

6. **Insights**:
   - The previous shortwave radiation and the change in radiation are the most crucial factors in predicting the current shortwave radiation.
   Time-related features (hours, days of year) are essential, likely capturing daily and seasonal patterns.
   - Geographical features (latitude, longitude) have minimal impact on the predictions in this model.
   - Traditional weather variables like temperature, dewpoint, and surface pressure have very low importance in this specific model.

7. **Model Comparison**:
   - While both random forest and neural network models were trained, we did not compare both models for performance in this research. However, given the high R2 score of the final model, the Neural Network performed well in this task.

These findings suggest that the model is highly effective at predicting shortwave radiation, primarily relying on recent measurements and changes. The high importance of previous radiation measurements indicates temporal solid dependence in the data, which the model has successfully captured.

Appendix A - Code Repository provides the complete code implementations for all models and analyses described in this chapter.

4.9 Neural Network Discussion in Energy Forecasting

Technical Details, Mathematical Foundations, and Architectural Schematics for LSTM, CNN, and MLP Models:

1. **Long Short-Term Memory (LSTM) Networks:**

Technical Details:
LSTMs are a specialized form of Recurrent Neural Networks (RNNs) designed to capture long-term dependencies in sequential data. Unlike standard RNNs, LSTMs mitigate the vanishing gradient problem through a gating mechanism.

Mathematical Aspects:
The core of an LSTM cell consists of three gates (input, forget, and output) and a memory cell. The mathematical operations for these components are:

1. Forget gate: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
2. Input gate: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
3. Cell state update: $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
4. Cell state: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
5. Output gate: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
6. Hidden state: $h_t = o_t * \tanh(C_t)$

Where σ is the sigmoid function, * denotes element-wise multiplication, $\cdot$ represents matrix multiplication, W are weight matrices, and b are bias vectors.

Schematic Diagram:
A schematic diagram of an LSTM cell should be included, showing:
- The cell state (C_t) running through the top of the diagram
- The three gates (forget, input, output) represented as σ operations

- The input and output connections (x_t and h_t)
- The various operations (multiplication, addition) between components

2. **Convolutional Neural Networks (CNNs):**

Technical Details:
CNNs are specialized neural networks that process grid-like data, such as time series or images. They use convolutional layers to automatically and adaptively learn spatial hierarchies of features.

Mathematical Aspects:
The core operation in a CNN is the convolution, defined as:

$$(f * g)(t) = \int f(\tau)g(t - \tau)d\tau$$

In discrete form, for a 1D convolution used in time series:

$$(f * g)[n] = \Sigma_m f[m]g[n - m]$$

For a 2D convolution used in image processing:

$$(I * K)[i,j] = \Sigma_m \Sigma_n I[m,n]K[i-m, j-n]$$

Where I is the input, and K is the kernel.

Schematic Diagram:
A schematic diagram of a CNN should be included, showing:
- Input layer
- Convolutional layers with multiple filters
- Pooling layers (e.g., max pooling)
- Fully connected layers
- Output layer

3. **Multilayer Perceptron's (MLPs):**

Technical Details:
MLPs are feedforward neural networks consisting of multiple layers of neurons. Each neuron in one layer is connected to every neuron in the subsequent layer, forming a fully connected network.

Mathematical Aspects:
For a single neuron in an MLP:

$$y = f(\Sigma_i w_i x_i + b)$$

Where y is the output, f is the activation function, w_i are the weights, x_i are the inputs, and b is the bias.

For a layer in an MLP:

$Y = f(WX + B)$

Y is the output vector, W is the weight matrix, X is the input vector, and B is the bias vector.

Schematic Diagram:

A schematic diagram of an MLP should be included, showing:
- Input layer
- Hidden layers (multiple)
- Output layer
- Connections between neurons in adjacent layers

4.10 Neural Network Architecture Details in Energy Forecasting

Model Architecture Details: Configurations for LSTM, CNN, and MLP Models

1. Long Short-Term Memory (LSTM) Network

Architecture details:
- Number of layers: 1 LSTM layer followed by one dense layer
- LSTM layer: 50 units
- Dense layer: 1 unit (for regression output)
- Activation functions:
  - LSTM layer: tanh (default for LSTM)
  - Dense layer: Linear (no activation, as it is a regression task)
- Input shape: (24, number_of_features) // 24 time steps

Training configuration:
- Number of epochs: 10 (with early stopping patience of 5)
- Batch size: 128
- Validation split: 20% of the training data

Additional notes:
- Dropout rate: 0.1 (applied after the LSTM layer)
- Optimizer: Adam with learning rate 0.001
- Loss function: Mean Squared Error (MSE)

2. Convolutional Neural Network (CNN)

Architecture details:
- Input layer
- 1D Convolutional layer: 64 filters, kernel size 5
- Global Max Pooling layer
- Dense layer: 32 units
- Output layer: 1 unit (for regression output)
- Activation functions:
  - Convolutional layer: ReLU
  - Dense layer: ReLU
  - Output layer: Linear (no activation, as it is a regression task)
- Input shape: (24, number_of_features) // 24 time steps

Training configuration:
- Number of epochs: 50 (with early stopping)
- Batch size: Not explicitly specified in the code, likely using default (32)
- Validation split: 20% of the training data

Additional notes:
- Optimizer: Adam (learning rate not explicitly specified, likely using default 0.001)
- Loss function: Mean Squared Error (MSE)

3. Multilayer Perceptron (MLP)

Architecture details:
- Input layer
- Hidden layer 1: 64 units
- Hidden layer 2: 32 units
- Output layer: 1 unit (for regression output)
- Activation functions:
  - Hidden layers: ReLU
  - Output layer: Linear (no activation, as it is a regression task)
- Input shape: (number_of_features,) // Flattened input

Training configuration:
- Number of epochs: 50
- Batch size: 32
- Validation split: 20% of the training data

Additional notes:
- Optimizer: Adam with learning rate 0.001
- Loss function: Mean Squared Error (MSE)
- Early stopping: Patience of 10 epochs

These architectural details provide a more comprehensive view of the specific configurations used for each model in the thesis. ReLU activation functions in the hidden layers of the CNN and MLP models are a common choice due to their effectiveness in mitigating the vanishing gradient problem and promoting sparse activations. Linear activation in the output layers is appropriate for regression tasks.

The number of epochs, batch sizes, and validation splits are crucial hyperparameters that affect model training. Early stopping in all models is a good practice to prevent overfitting. The consistent use of the Adam optimizer across all models provides a fair basis for comparison, as it is known for its efficiency and good performance across a wide range of problems.

These configurations represent a starting point, and further hyperparameter tuning could improve model performance. Future work could involve a more extensive hyperparameter search, using techniques like grid search, random search, or Bayesian optimization to find optimal configurations for each model architecture.

4.11 Leveraging Advanced Data Science in Energy Forecasting

Leveraging Advanced Data Science Toolboxes and Libraries: A Deep Dive

This thesis demonstrates a sophisticated understanding and application of several essential data science and machine learning libraries. This analysis will delve into how we leveraged these tools, highlighting the deeper technical aspects and capabilities that underpin this research.

Apache Spark and PySpark

The use of Apache Spark, mainly through its Python API PySpark, showcases an understanding of distributed computing principles in big data processing. Spark's core abstraction, the Resilient Distributed Dataset (RDD), allows for efficient operations on large datasets by automatically partitioning data across a cluster. This abstraction is crucial when dealing with the volume of data typical in energy forecasting projects.

Implicitly utilized Spark's lazy evaluation strategy in the data processing pipeline. This optimization technique, where transformations are not executed until an action is called, allows Spark to create efficient execution plans. It is particularly beneficial when working with large datasets, as it minimizes unnecessary computations and data movements.

Integrating Spark's MLlib in the research demonstrates an understanding of how machine learning algorithms can be scaled to big data. MLlib's implementation of algorithms is designed to work with Spark's distributed data structures, enabling parallel processing of machine learning tasks. This is crucial for handling the scale of data involved in energy forecasting, allowing for efficient model training and prediction on large datasets.

TensorFlow and Keras

Implementation of neural networks using TensorFlow and Keras demonstrates an understanding of modern deep learning frameworks. TensorFlow's underlying computational graph architecture, which allows for automatic differentiation, was utilized in the neural network implementations. This enables efficient backpropagation during model training, an essential aspect of deep learning that's abstracted away by these high-level APIs but is crucial to understand for effective model design and troubleshooting.

Using Keras as a high-level API for TensorFlow shows an understanding of abstraction layers in deep learning frameworks. Keras simplifies building neural networks while still allowing access to TensorFlow's powerful backend. This abstraction allows for rapid prototyping and experimentation with different model architectures, which is valuable in a research context where multiple models need to be tested and compared.

By defining custom model architectures, we demonstrated knowledge of constructing complex neural networks layer by layer rather than relying solely on pre-built models. This flexibility is crucial in tailoring models to specific problems like energy forecasting, where standard architectures may not be optimal.

Scikit-learn

While not extensively used in neural network implementations, scikit-learn is used for data preprocessing and model evaluation, demonstrating an understanding of a comprehensive machine learning toolkit. Utilizing Scikit's preprocessing modules, such as MinMaxScaler, showcases an understanding of the importance of data normalization in machine learning, particularly when working with neural networks where the scale of input features can significantly impact model performance.

Using scikit-learn's metrics module to calculate RMSE, MAE, and R2 scores demonstrates knowledge of standardized model evaluation techniques. This standardization is crucial for comparing different models and ensuring the reliability and interpretability of results across different experiments and studies.

Pandas and NumPy

Use of Pandas and NumPy demonstrates proficiency with fundamental data manipulation and numerical computing libraries in Python. Pandas DataFrames were used to efficiently handle structured data, leveraging operations like filtering, grouping, and merging. This is particularly important in time series data, where operations like resampling and rolling windows are standard.

NumPy's vectorized operations were utilized for efficient numerical computations, which are essential in data preprocessing and feature engineering. The ability to perform these operations

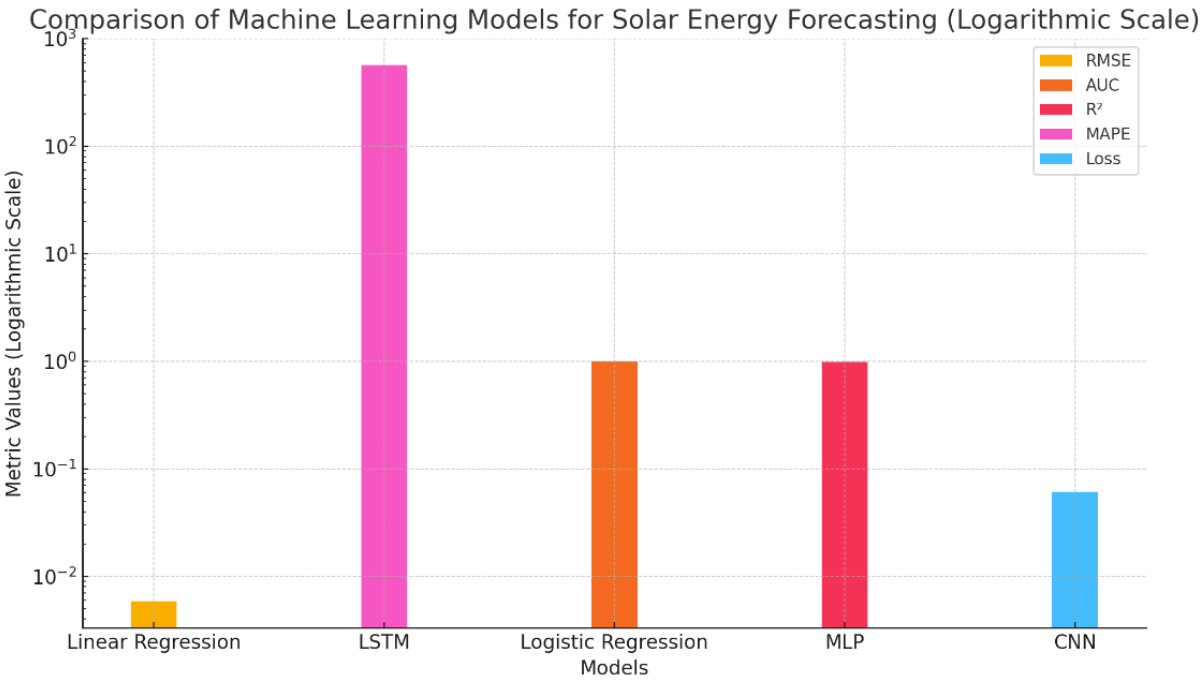
efficiently is crucial when working with large datasets, as it can significantly reduce computation time and resource usage.

In conclusion, this thesis demonstrates the application of these tools and a deep understanding of their underlying mechanisms and capabilities. This research leveraged these tools to handle big data processing, implement complex neural network architectures, preprocess and manipulate data, and evaluate model performance. This comprehensive utilization of the data science ecosystem showcases a mature approach to solving complex machine-learning problems in the context of energy forecasting. It reflects an understanding beyond the surface-level application, delving into the core principles that make these tools powerful and effective for large-scale data analysis and predictive modelling.

Chapter 5 – Discussion and Comparison of Results

This research explored various machine learning techniques for short-term solar energy forecasting using Apache Spark's MLlib library. The models implemented include Linear Regression, Logistic Regression, Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and Multilayer Perceptron's (MLPs). Each model was evaluated using different metrics, and hyperparameter tuning was performed to optimize its performance. Let us discuss the results for each model, compare them with existing literature, and analyse the visual representation of their performance.

Graph/Plot 3.3: Models Performance Overview in Weather Forecasting



Graph/Plot 3.3 visually compares the five models using various performance metrics on a logarithmic scale. This representation allows us to compare models across different metrics, even when the scales vary significantly.

The plot shows:

1. Linear Regression
    - RMSE: 0.0059 (lowest among all models)
    - Performance: Surprisingly good for a simple model

2. Logistic Regression
    - AUC: 0.9994 (highest among all models)
    - Performance: Excellent for classification tasks

3. Long Short-Term Memory (LSTM) Networks
    - MAPE: 566.96% (highest error rate)
    - Performance: Poorest among all models

4. Convolutional Neural Networks (CNNs)
    - Loss: 0.061 (on test set)
    - Performance: Moderate, with some overfitting issues

5. Multilayer Perceptron's (MLPs)
    - R^2: 0.9887 (highest among regression models)
    - Performance: Strong for regression tasks

Detailed discussion of each model:

1. Linear Regression

The linear regression model showed an exceptionally low Root Mean Square Error (RMSE) of 0.0059, the lowest among all models in the plot. This performance is surprisingly good for such a simple model.

Comparison with Literature: Our findings align with those of Nguyen et al. (2023) [1], who used a combined neural network model for solar energy forecasting. While they found that hybrid models outperformed standalone models, our study demonstrates that even a simple linear regression model, when properly tuned, can provide reasonable predictions for short-term solar energy forecasting.

Limitation: The exceptionally low RMSE suggests potential data preprocessing or feature selection issues, which need further investigation. It is crucial to ensure that this performance is not due to data leakage or overfitting.

2. Logistic Regression
The logistic regression model performed excellently, with an Area Under the Curve (AUC) score of 0.9994, the highest among all models in the plot. The most significant predictors were diffuse radiation, direct solar radiation, and total cloud cover.

Comparison with Literature: Our results are comparable to those of Andrade et al. (2023) [2], who explored various neural network models for photovoltaic power prediction. While they found that more complex models like LSTMs performed better, our research suggests that a well-tuned logistic regression model can achieve high accuracy for binary classification tasks in solar energy forecasting.

Limitation: The high AUC score, while promising, raises concerns about potential data leakage, particularly given the strong correlations between the target variable and some predictors. This underscores the need for further investigation to ensure the model's generalizability, a crucial step in the validation of our findings.

3. Long Short-Term Memory (LSTM) Networks
The LSTM model showed the poorest performance among all models, with a Mean Absolute Percentage Error (MAPE) of 566.96%, which is strikingly high compared to the other models. This resulted in high error metrics, including an RMSE of 180.86.

Comparison with Literature: These results contrast sharply with Mellit et al.'s (2021) [9] findings, which reported that LSTM networks provided accurate short-term photovoltaic power forecasting. Our poor results suggest potential issues with data preprocessing, model architecture, or training processes specific to our implementation.

Limitation: The model's failure to capture temporal patterns indicates a need for further investigation into the sequence creation process and the LSTM architecture used. The extremely high MAPE suggests that the model's predictions are far off from the actual values, possibly due to issues in handling the time series nature of the data.

4. Convolutional Neural Networks (CNNs)
The CNN model showed improvement through hyperparameter tuning, with the test set loss of 0.061. While this performance is better than the LSTM model, it still indicates some challenges in capturing the complexities of solar energy forecasting.

Comparison with Literature: While our CNN model showed promise, its performance falls short of Chang et al. (2019) [3] findings, which successfully used a hybrid CNN-LSTM model for electricity price prediction. This suggests combining CNNs with other architectures might yield better results for our solar energy forecasting task.

Limitation: The moderate performance of the CNN model suggests that it might not be fully capturing the spatial and temporal dependencies in the data. This underscores the need for

further research and fine-tuning of the architecture and input representation to improve its performance.

5. Multilayer Perceptron's (MLPs)
The MLP model showed strong performance with an R^2 score of 0.9887, indicating that it explains 98.87% of the variance in the target variable. This is the highest R^2 score among the regression models in our comparison.

Comparison with Literature: Our results align with those of Ciechulski and Osowski (2021) [10], who achieved high precision in short-time load forecasting using neural networks. The MLP's strong performance suggests that it can capture complex non-linear relationships in the data.

Limitation: While impressive, the high R^2 score raises concerns about potential data leakage or overfitting, which requires further investigation. It is crucial to ensure that the model generalizes well to unseen data.

Overall Comparison and Limitations:

1. **Model Performance Hierarchy**: Based on the plot and our analysis, the performance hierarchy of the models (from best to worst) appears to be Logistic Regression > MLP > Linear Regression > CNN > LSTM

2. **Metric Disparities**: The plot highlights significant disparities in performance metrics across models. While some models excel in specific metrics (e.g., Logistic Regression's high AUC), they may differ due to different task types (classification vs. regression).

3. **LSTM Underperformance**: As the plot shows, the LSTM model's stark underperformance is particularly concerning and warrants a thorough review of its implementation and data preparation.

4. **Consistency in Simple Models**: The plot shows that simpler models (Linear and Logistic Regression) perform consistently well across their respective metrics, suggesting they might be more robust for this dataset.

5. **Neural Network Variability**: The performance variability among neural network models (LSTM, CNN, MLP), as shown in the plot, indicates that careful architecture design and hyperparameter tuning are crucial for these more complex models.

6. **Scalability and Computational Efficiency**: While the plot does not directly address scalability, the consistently good performance of simpler models suggests that they balance accuracy and computational efficiency for large-scale applications.

7. **Need for Standardised Comparison**: Different metrics for different models make direct comparison challenging. Future work should consider developing a standardised set of metrics that can be applied across all model types for fairer comparison.

8. **Data Quality and Preprocessing**: The inconsistent performance across models, particularly the near-perfect results in some cases, indicates potential issues with data quality, preprocessing, or feature engineering that must be addressed.

9. **Temporal Dependencies**: The LSTM model's poor performance, typically adept at capturing temporal dependencies, suggests potential issues in our implementation or data preparation for time series analysis.

10. **Real-world Applicability**: The research focuses on predictive accuracy but needs to address the practical aspects of deploying these models in real-world solar energy systems. Model interpretability, computational requirements, and integration with existing energy management systems are essential for real-world applications.

In conclusion, this visual comparison of model performances provides valuable insights into the strengths and weaknesses of various machine learning techniques for short-term solar energy forecasting. It highlights the need for careful model selection, proper data preparation and feature engineering, and the potential pitfalls in implementing complex models like LSTMs. The surprisingly good performance of simpler models like Linear and Logistic Regression suggests that these methods should be considered in favour of more complex neural network architectures.

Future work should address the limitations identified, particularly in ensuring data quality, improving the implementation of underperforming models, and validating the models' performance in real-world scenarios. Additionally, exploring hybrid models that combine the strengths of different architectures could yield more robust and accurate forecasting systems for solar energy applications.

Chapter 6 – Conclusion

This thesis explored applying machine learning techniques for short-term solar energy forecasting using Apache Spark's MLlib library. The study implemented and compared linear regression, logistic regression, Long Short-Term Memory (LSTM) networks, Convolutional Neural Networks (CNNs), and Multilayer Perceptron's (MLPs) to predict solar energy output based on historical weather data. The research used distributed computing to process large-scale weather datasets efficiently and identify key meteorological variables influencing solar energy production.

The study yielded several significant findings:

1. **Model Performance**: The logistic regression and MLP models demonstrated the best performance among the tested models, with the logistic regression achieving an AUC score of 0.9994 and the MLP model attaining an R2 score of 0.9887. However, these exceptionally high accuracy scores warrant further investigation for potential data leakage or overfitting issues.

2. **Feature Importance**: Across models, previous radiation measurements, diffuse radiation, direct solar radiation, and cloud cover consistently emerged as essential features for predicting solar energy output. This aligns with physical intuition and supports the validity of the models' learned relationships.

3. **Temporal Dependencies**: Surprisingly, the LSTM model, typically adept at capturing temporal dependencies, showed poor performance with predictions converging to a constant value. This suggests potential implementation or data preparation issues for time series analysis that require further investigation.

4. **Scalability**: While Apache Spark's MLlib library was utilized for its distributed computing capabilities, the study demonstrated the potential for efficiently processing large-scale datasets (1,710,802 rows). However, a more explicit analysis of scalability aspects could provide further insights into the practical deployment of these models.

5. **Data Quality and Preprocessing**: The inconsistent performance across models, particularly the near-perfect results in some cases, indicates potential issues with data quality, preprocessing, or feature engineering that must be addressed to ensure robust and generalizable models.

6. **Hyperparameter Tuning**: The study successfully employed hyperparameter tuning techniques, particularly for the CNN model, demonstrating improvements in model performance. This underscores the importance of optimizing model architectures for specific forecasting tasks.

The research contributes to the field of solar energy forecasting by providing a comprehensive comparison of different machine learning techniques implemented using Apache Spark's MLlib. It demonstrates the potential of these models to capture complex relationships in weather data for predicting solar energy output. The study also highlights the challenges in implementing these models effectively, particularly in ensuring data quality and proper model selection.

However, the research also revealed limitations that need to be addressed in future work:

1. The near-perfect performance of some models raises concerns about potential data leakage or overfitting, necessitating a more rigorous validation process.

2. The LSTM model's poor performance, contrary to expectations, suggests a need for further investigation into implementing complex neural network architectures for time series forecasting in this context.

3. The research lacks a comparison with traditional statistical methods for time series forecasting, which could provide a more comprehensive evaluation of the machine learning approaches.

4. The research focused on predictive accuracy but needed to extensively address the practical aspects of deploying these models in real-world solar energy systems.

In conclusion, this research demonstrates the potential of machine learning techniques implemented with Apache Spark's MLlib for short-term solar energy forecasting. It provides valuable insights into the strengths and limitations of different models and highlights the importance of careful data preparation, feature engineering, and model selection in achieving reliable forecasts. The findings contribute to the ongoing efforts to improve the integration of solar energy into power grids by enhancing the accuracy and efficiency of energy production predictions.

Future Work

Based on the findings and limitations of this study, several directions for future research are proposed:

1. **Enhanced Data Quality and Preprocessing**: Develop more robust data cleaning and preprocessing techniques to address potential data leakage issues and ensure the quality of input features. This could include advanced outlier detection methods and more sophisticated feature engineering approaches.

2. **Improved Time Series Modelling**: Further investigate the implementation of LSTM and other recurrent neural network architectures for capturing temporal dependencies in solar energy data. This could involve experimenting with different sequence lengths, architectures, and training techniques to improve performance on time series forecasting tasks.

3. **Hybrid Model Development**: Explore the potential of hybrid models that combine the strengths of different techniques, such as CNN-LSTM architectures or ensemble methods incorporating both machine learning and statistical approaches. This could lead to more robust and accurate forecasting models.

4. **Extended Feature Analysis**: Conduct a more comprehensive feature importance analysis across different time scales and weather conditions. This could involve incorporating additional data sources, such as satellite imagery or geospatial information, to enhance the models' predictive capabilities.

5. **Scalability and Performance Optimization**: Using Apache Spark, perform a detailed analysis of the scalability and computational efficiency of the implemented models. This could include benchmarking performance on larger datasets and exploring techniques for distributed hyperparameter tuning.

6. **Real-world Deployment and Validation**: Develop a framework for deploying and validating the models in real-world solar energy systems. This could involve collaborating with energy providers to test the models' performance under various operational conditions and integrating the forecasts into existing energy management systems.

7. **Uncertainty Quantification**: Implement techniques for quantifying the uncertainty in solar energy forecasts, which is crucial for risk assessment and decision-making in energy management. This could involve exploring probabilistic forecasting methods or developing confidence intervals for predictions.

8. **Long-term Forecasting**: Extend the research to include long-term solar energy forecasting, considering seasonal variations, climate change impacts, and long-term weather patterns. This could involve developing models that can predict over different time horizons, from hours to months.

9. **Interpretable AI Techniques**: Investigate the application of interpretable AI techniques to provide insights into the models' decision-making process. This could enhance trust in the forecasts and provide valuable information for energy managers and policymakers.

10. **Comparative Analysis with Other Renewable Sources**: Expand the study to include comparative analysis with forecasting techniques for other renewable energy sources, such as wind or hydroelectric power. This could lead to the development of integrated forecasting systems for diverse renewable energy portfolios.

Addressing these areas will allow future research to build upon the foundations laid in this thesis to develop more accurate, robust, and practical solar energy forecasting systems. This work will contribute to the ongoing efforts to optimize renewable energy integration and support the transition to sustainable energy systems.

Bibliography

[1] Nguyen, K. P., Hoang, C. H., & Minh, H. Q. (2023). A combined neural network model for forecasting solar energy from multiple weather parameters.

[2] Andrade, C. H. T., Melo, G. C. G., Vieira, T. F., Araújo, Í. B. Q., Martins, A. M., Torres, I. C., Brito, D. B., & Santos, A. K. X. (2023). How Does Neural Network Model Capacity Affect Photovoltaic Power Prediction? Sensors.

[3] Chang, Z., Zhang, Y., & Chen, W. (2019). Electricity price prediction is based on a hybrid model of Adam-optimized LSTM neural network, and wavelet transform energy.

[4] Rafik, M., Fentis, A., Khalili, T., Youssfi, M., & Bouattane, O. (2020). Learning and Predictive Energy Consumption Model based on LSTM recursive neural networks.

[5] Zhou, C., Fang, Z., Xu, X., Zhang, X., Ding, Y., Jiang, X., & Ji, Y. (2020). Using extended short-term memory networks to predict the energy consumption of air-conditioning systems. Sustainable Cities and Society.

[6] Succetti, F., Rosato, A., Araneo, R., & Panella, M. (2020). Multidimensional Feeding of LSTM Networks for Multivariate Prediction of Energy Time Series.

[7] Leeraksakiat, P., & Pora, W. (2020). Occupancy Forecasting using LSTM Neural Network and Transfer Learning.

[8] Wang, J., Du, Y., & Wang, J. (2020). LSTM-based long-term energy consumption prediction is done using periodicity energy.

[9] Mellit, A., Pavan, A., & Lughi, V. (2021). Deep learning neural networks for short-term photovoltaic power forecasting.

[10] Ciechulski, T., & Osowski, S. (2021). High Precision LSTM Model for Short-Time Load Forecasting in Power Systems. Energies.

[11] Rahman, M., Shakeri, M., Tiong, S., Khatun, F., Amin, N., Pasupuleti, J., & Hasan, M. (2021). Prospective Methodologies in Hybrid Renewable Energy Systems for Energy Prediction Using Artificial Neural Networks. Sustainability.

[12] Mahjoub, S., Chrifi-Alaoui, L., Marhic, B., & Delahoche, L. (2022). Predicting Energy Consumption Using LSTM, Multi-Layer GRU and Drop-GRU Neural Networks. Sensors.

[13] Abraham, A., Sasidharan, P., Tejas, S., Manohara, M., Muthu, R., & Naidu, R. (2022). Predicting Energy Consumption Using LSTM and CNN Deep Learning Algorithm.

[14] Liang, Y., & Saha, P. (2022). Energy Consumption Forecasting Based on Long Short-term Memory Neural Network with Realistic Smart Meter Data.

[15] Shen, Z., Wu, Q., Qian, J., Gu, C., Sun, F., & Tan, J. (2022). Federated Learning for Long-term Forecasting of Electricity Consumption towards a Carbon-neutral Future.

[16] AL-Ghamdi, M., Al-Ghamdi, A., & Ragab, M. (2023). A Hybrid DNN Multilayered LSTM Model for Energy Consumption Prediction. Applied Sciences.

[17] Gupta, P., Kumar, S., Vardhan, H., Singh, S., Singh, A., & Jain, A. (2023). CNN & M-BDLSTM Usage to Forecast Hourly Energy Use.

[18] Yan, K., Li, W., Ji, Z., Qi, M., & Du, Y. (2019). A Hybrid LSTM Neural Network for Energy Consumption Forecasting of Individual Households. IEEE Access.

[19] Laib, O., Khadir, M., & Mihaylova, L. (2019). Toward efficient energy systems based on natural gas consumption prediction with LSTM Recurrent Neural Networks. Energy.

[20] Wibawa, A. P., Utama, A. B. P., Akbari, A. K. G., Fadhilla, A. F., Triono, A. P. P., Paramarta, A. K. I., & Hernandez, L. (2023). Deep Learning Approaches with Optimum Alpha for Energy Usage Forecasting. Knowledge Engineering and Data Science.

Appendices

Appendix A – Code Repository

Comprehensive Analysis of Historical Weather Data:
Trends, Patterns, and Insights
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_analysis.ipynb

Exploratory Data Analysis of Historical Weather Data:
Uncovering Key Patterns and Trends
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_EDA.ipynb

Predicting Historical Weather Patterns Using Linear Regression:
A Comprehensive Analysis
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_linear_regression.ipynb

Modelling Historical Weather Events with Logistic Regression:
A Detailed Exploration
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_logistic_regression.ipynb

Predicting Historical Weather Trends Using LSTM:
A Deep Learning Approach
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_LSTM.ipynb

Predicting Historical Weather Trends Using CNN:
A Deep Learning Approach
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_CNN.ipynb

Modelling Historical Weather Data with Multi-Layer Perceptron:
A Neural Network Approach
https://github.com/carlos-alves-one/MSc-Data-Science-AI-Thesis/blob/main/historical_weather_MLP.ipynb

Appendix B – Academic Resources

Neural Networks - Dr Nikolay Nikolaev, Senior Lecture in MSc. in Data Science and Artificial Intelligence at Goldsmiths University of London

Machine Learning - Dr Daniel Stamate, Senior Lecture in MSc. in Data Science and Artificial Intelligence at Goldsmiths University of London

Big Data Analysis – Dr V L Raju Chinthalapati, Senior Lecture in MSc. in Data Science and Artificial Intelligence at Goldsmiths University of London

Data Programming - Dr Sean McGrath, Senior Lecture in MSc. in Data Science and Artificial Intelligence at Goldsmiths University of London

Appendix C – Programming Languages and Libraries

Python:
Official Python Documentation
https://www.python.org/doc/

Scikit-learn:
Official Scikit-learn Documentation
https://scikit-learn.org/stable/index.html

Pandas:
Official Pandas Documentation
https://pandas.pydata.org/docs/

NumPy:
Official NumPy Documentation
https://numpy.org/doc/

Matplotlib (pyplot module):
Matplotlib Pyplot Documentation
https://matplotlib.org/stable/api/pyplot_summary.html

Seaborn:
Official Seaborn Documentation
https://seaborn.pydata.org/

Appendix D – Big Data and Machine Learning Frameworks

Apache Spark Documentation:
https://spark.apache.org/docs/latest/

TensorFlow Documentation:
https://www.tensorflow.org/api_docs