Title of Dissertation:

A socio-technical framework for forecast-informed optimisation of public electric vehicle charging under uncertainty: a case study of Perth and Kinross Council, UK

By

Student Number 2229417

A Dissertation submitted in partial fulfilment of the requirements for the degree of

MSc Big Data and Digital Futures

Supervised by DR KAVIN NARASIMHAN

University of Warwick

Centre for Interdisciplinary Methodologies

August 2025

Abstract

Unmanaged public Electric Vehicle (EV) charging creates volatile demand, straining local power grids and resulting in high operational costs for local authorities. While extensive research exists on forecasting EV energy consumption, as well as those optimising charging schedules, most studies rely on deterministic, single-point predictions that do not account for real-world unpredictability. This dissertation bridges that gap by integrating these two pillars – forecasting and optimisation – within a framework that explicitly manages the uncertainty inherent in user behaviour. To achieve this, I develop and validate a four-stage Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF), which begins with a socio-technical analysis and culminates in a cost-minimising linear program.

Using a dataset of 55,878 charging sessions from Perth and Kinross Council, UK (2016-2019), I first tested five forecasting models to find the most accurate point forecast. The XGBoost model performed best (MAE = 15.83 kWh, $R^2 = 0.64$) and was subsequently adapted to perform probabilistic forecasting via quantile regression. These probabilistic forecasts were then used to develop an optimised schedule using linear programming. Finally, this "Forecast-Informed" strategy was compared against two baselines: "Uncontrolled" charging and a simple "Rule-Based" window.

The forecast-informed strategy delivered substantial economic benefits. On a sample day, it reduced total costs by £994.12 (53.2%) as a single-day high. Over a typical week, the savings were £1,602.36 (41.7%), and over a month, they reached £3,953.09 (38.0%), with savings primarily driven by mitigating expensive peak demand charges. While these results should be viewed with caution due to the age of the data and simplified tariffs, sensitivity analyses confirmed that the savings are consistent across different price scenarios. The finding is that treating unpredictability as a key factor makes it possible to create cheaper, more reliable charging schedules that stay within operational limits, offering a valuable framework for policymakers and infrastructure operators.

Keywords: Electric Vehicle (EV) Charging; Demand Forecasting; Uncertainty Quantification; Machine Learning; Linear Programming; Cost Optimisation; Socio-Technical Systems

Table of contents

List of Tables

List of Figures

Acknowledgements

I want to express my sincere gratitude to the Centre for Interdisciplinary Methodologies (CIM) for awarding me the CIM Master's Scholarship. This research would not have been possible without this generous financial support for my study.

I am grateful to my supervisor, Dr. Kavin Narasimhan, for her invaluable guidance, detailed feedback, and unwavering support throughout the dissertation process. Her insights were instrumental in shaping the project's direction. I would also like to extend my thanks to my personal tutor, Dr. Michael Castelle, for his insights throughout my program.

My appreciation also goes to all the module convenors and tutors at CIM for their excellent teaching.

Finally, I would like to thank my family for their immense and unwavering support throughout my studies. Their encouragement has been a source of great strength.

Academic Integrity Declaration – CIM Dissertation

In submitting my work, I confirm that:

1. I have read the guidance on academic integrity provided in the Student Handbook and understand the University regulations in relation to Academic Integrity. I am aware of the potential consequences of Academic Misconduct.
2. I declare that the work is all my own, except where I have stated otherwise.
3. No substantial part(s) of the work submitted here has also been submitted by me in other credit bearing assessments courses of study (other than in certain cases of a resubmission of a piece of work), and I acknowledge that if this has been done this may lead to an appropriate sanction.
4. Where a generative Artificial Intelligence such as ChatGPT has been used I confirm I have abided by both the University guidance and specific requirements as set out in the Student Handbook and the Assessment brief. I have clearly acknowledged the use of any generative Artificial Intelligence in my submission, my reasoning for using it and which generative AI (or AIs) I have used. Except where indicated the work is otherwise entirely my own.
5. I understand that should this piece of work raise concerns requiring investigation in relation to any of points above, it is possible that other work I have submitted for assessment will be checked, even if marks (provisional or confirmed) have been published.
6. Where a proof-reader, paid or unpaid was used, I confirm that the proof-reader was made aware of and has complied with the University's proofreading policy.
7. I consent that my work may be submitted to Turnitin or other analytical technology. I understand the use of this service (or similar), along with other methods of maintaining the integrity of the academic process, will help the University uphold academic standards and assessment fairness.

Wordcount (Main text): 10,987

Chapter 1: Introduction

1.1 Background and Context

The United Kingdom (UK) has committed to achieving net-zero greenhouse gas emissions by 2050 (Ullah et al., 2022). With domestic transport being the largest source of these emissions, the government has positioned electric vehicles (EVs) as a cornerstone of its decarbonisation strategy – mandating a phase-out of new petrol and diesel vehicle sales by 2035 (GOV.UK, 2025; Logan et al., 2021). The adoption of EVs offers a pathway to sustainability through reduced point-of-use emissions – emissions generated directly by vehicles during operation (Rashid et al., 2024).

However, this rapid adoption has placed significant strain on public EV charging stations, the infrastructure intended to support this transition (Krishna et al., 2024). It has accelerated the demand on local electricity grids, creating risks of destabilising load spikes, higher operational costs, and grid instability (Singh et al., 2024; Vishnu et al., 2023). These issues present critical operational and financial challenges for local authorities responsible for delivering equitable EV access (Hopkins et al., 2023). They must ensure the operational efficiency of charging infrastructures and make sure they serve all communities – from dense urban centres to remote rural areas, thereby preventing the emergence of "charging deserts," regions with limited access to charging stations.

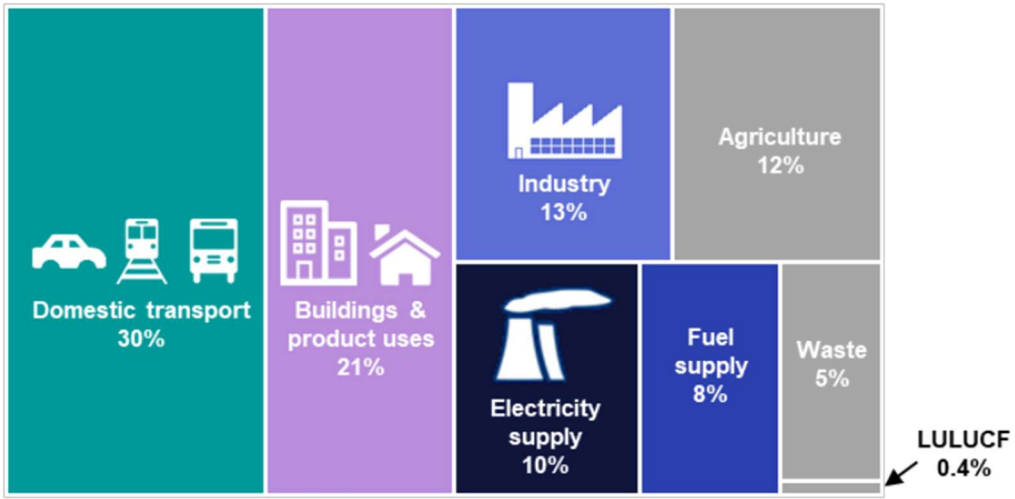


Figure 1: UK provisional GHG emissions for 2024 by sector

(*Land Use, Land-Use Change, and Forestry (LULUCF*)

(Source: GOV.UK, 2025)

1.2 Problem Statement and Motivation

Perth and Kinross Council (PKC) exemplifies the financial pressures facing local authorities. A 2022 report from its Head of Planning & Development, authored by Deans & Moran (2022), reveals that PKC initially incentivised EV adoption through free public charging, but rising electricity and maintenance costs have made this model unsustainable. It showed that electricity and support costs doubled from £120,000 in 2021/22 to a projected £230,000 for 2022/23. With the Scottish Government's initial capital grants now expired, PKC is solely responsible for these expenses.

PKC's situation is an indication of a broader issue: the prevalence of unmanaged EV charging – where vehicles draw power immediately upon connection – causes unpredictable demand spikes that strain local grids and inflate costs, especially when they coincide with peak-period electricity tariffs (Chang et al., 2021; Hopkins et al., 2023). While PKC's responsive move to an energy-based tariff from 2023 is intended to manage costs, it does not address the underlying problem of volatile, unpredictable, and unmanaged demand.

To address this, methods for estimating charging demand, such as statistical regression models and time-series analysis, have been applied. However, their performance can be limited when capturing complex, non-linear patterns (Koohfar et al., 2023). Machine learning (ML) methods, including artificial intelligence (AI), provide new opportunities, through an improved ability to estimate charging demands (Rashid et al., 2024). Nevertheless, current deterministic forecasting models (models that estimate outcomes without incorporating uncertainties) can be inadequate in accounting for real-world behavioural uncertainties, such as spontaneous travel patterns and unforeseen local events, creating risk of failure in the real world, which can lead to over-provision of costly energy or an under-supply (Akshay et al., 2024; Fan et al., 2023). This gap highlights the need for robust, uncertainty-aware forecasting methodologies (methods that quantify and incorporate a range of possible variations in their estimation), thereby enabling optimised, cost-effective, and reliable management of EV charging infrastructure.

1.3 Research Objectives and Questions

The primary focus of this research is to enhance the operational efficiency and economic viability of public EV charging stations. To achieve this, my research proposes and evaluates a Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF). The research is guided by a primary research question (RQ1) concerning the benefits of this framework, which

is supported by a secondary question (RQ2) on forecasting accuracy. The research questions and supporting hypotheses are as follows:

RQ1: What cost-saving benefits can be achieved by implementing the ST-UOF compared to uncontrolled and rule-based charging strategies?

- **H1:** The forecast-informed optimisation strategy substantially reduces electricity costs compared to uncontrolled and simple rule-based charging methods.

RQ2: To enable this optimisation, can machine learning models (SVR, XGBoost, CNN-LSTM) outperform traditional statistical models (MLR, SARIMA) in forecasting short-term EV charging demand?

- **H2:** ML models significantly improve forecasting accuracy over traditional statistical approaches, providing a reliable input for the optimisation framework.

1.4 Novelty and Contribution

This research contributes to socio-technical transition theory by examining the interplay between technical forecasting methods, optimisation, and social equity within infrastructure transitions. Socio-technical transition theory provides a framework for understanding how large-scale systems, such as transportation and energy networks, undergo long-term change through the interaction of technology, policy, and social practices (Geels, 2019). While many studies rely on single-point, deterministic forecasts, this work moves beyond these limits by employing probabilistic forecasting techniques, which estimate and represent the range of possible outcomes in forecasting models (Bracale et al., 2020), thereby accounting for uncertainty. Using techniques such as Quantile Regression, my approach captures the spectrum of potential EV charging demand rather than a single predicted value (Abdar et al., 2021; Cao et al., 2024). Also, to bridge the gap between theoretical modelling and practical application, the methodology is empirically tested using real-world data from the PKC public EV charging network. Furthermore, the analysis provides tangible evidence by comparing energy cost outcomes across different charging strategies: uncontrolled, controlled, and forecast-informed. Ultimately, the research offers insight into a framework that can help infrastructure operators and policymakers enhance the financial viability and operational reliability of public EV charging stations, while speaking to equity considerations in access.

1.5 Policy and Industry Relevance

The findings from this research offer actionable insights for key stakeholders. For local authorities like PKC, it provides a practical blueprint for managing charging networks more efficiently, easing budgetary pressures while improving service quality. At the national level, these intelligent, forecast-informed strategies support objectives for equitable and effective deployment of EV infrastructure. Furthermore, the methodological approach provides a basis for industry actors to develop or improve commercial smart charging solutions. This study serves as a foundation for these applications, as its scope and resources limit deeper exploration.

This dissertation aligns with strategic initiatives like the UK Government's "EV Smart Charging Action Plan" and the Scottish Government's "Switched on Towns and Cities Challenge Fund," both promoting interoperable, data-driven, user-centred EV infrastructure (GOV.UK, 2023; Trust, 2019). By providing empirical evidence and practical tools, this dissertation seeks to inform the implementation of these policy objectives and support the development of a robust and equitable EV ecosystem.

1.6 Structure of the Dissertation

The dissertation is structured as follows: Chapter 2 critically evaluates existing literature, Chapter 3 details methodology and data selection, Chapter 4 presents empirical findings from exploratory, forecasting and optimisation analyses, Chapter 5 critically discusses findings within broader socio-technical implications, explicitly acknowledging research constraints, limitations, and outlines recommendations for future research, and Chapter 6 concludes by summarising key insights of the research.

Chapter 2: Literature Review

The previous chapter established the aim of this study. This chapter builds the theoretical and methodological foundation by critically evaluating the relevant literature. The review establishes the socio-technical context of EV infrastructure, showing why new solutions are needed. It then examines the progression of research in forecasting and optimisation to identify effective technical approaches. By synthesising these distinct but interconnected fields, it identifies a critical research gap and formulates the specific contribution of this dissertation.

2.1 Socio-Technical Dimensions of Public Charging

The technical design of charging systems is constrained by travel patterns and infrastructure distribution (Efthymiou et al., 2015). Consequently, the theoretical foundation for this work begins with the socio-technical setting. Geels (2019) argues that large-scale infrastructure change is rarely a simple technological substitution, but a complex interaction of technology, policy, and user behaviour that shapes outcomes like equity, cost, and grid stability (Sovacool et al., 2021).

The shift towards EVs exemplifies this infrastructure change. Bhatt et al. (2024) contend that it requires not only technical readiness but also regulatory and social adaptation, such as changes in charging routines, trip planning, and shared norms around charger etiquette and fair access. These behaviours determine when and where demand appears and, therefore, what the infrastructure must absorb. As Steinhilber et al. (2013) note, infrastructure deployment carries obvious social and political dimensions. For example, the UK's public charging network: the Committee of Public Accounts (2024) report shows that 43% of chargers are clustered in London and the South East. Khan et al. (2022) warn that this pattern fosters charging deserts in rural and disadvantaged areas, risking a two-tier system and deepening transport inequalities. Regions like PKC, combining accessible towns with remote settlements, tend to experience this pattern on a small scale.

Human behaviour further complicates matters. Mandolakani & Singleton (2024) and Quiros-Tortos et al. (2015) show that charging patterns vary by place and socio-demographic profile, ranging from short, frequent top-ups in urban settings to a reliance on long-distance rapid charging among rural motorists. Khan et al. (2022) also find that higher-income households are more likely to charge at home, shifting dependence on public infrastructure towards lower-income zones. To guide fairer access, Bedford et al. (2023) propose metrics such as average

drive-time to a charger, arguing that travel time captures functional access more faithfully than raw charger counts. It reflects road speeds and settlement patterns, exposes communities facing long trips without home charging, and provides authorities with a standard benchmark for urban and rural decisions. However, such static frameworks struggle to capture the live, hour-to-hour demand patterns that operational teams must manage (Manny et al., 2022); they can miss short surges and large queues, even in areas that appear well served on paper. Thus, equity-aware planning necessitates an operational layer that anticipates variability rather than relying on smooth averages.

Taken together, these issues present a central research challenge, identified by Steinhilber et al. (2013): how can demand be managed in a way that respects complex human patterns while maintaining operational reliability? These distributional and behavioural dynamics make demand estimation crucial for a credible operational strategy. Without forward estimates, operators cannot right-size capacity, avoid peaks, or set fair pricing rules (Ostermann & Haug, 2024).

2.2 The Evolution of EV Forecasting Methodologies

Ullah et al. (2022) and Vishnu et al. (2023) state that accurate forecasting enables intelligent charging by providing foresight over variable loads from uncoordinated charging. This is essential because control and scheduling decisions are made in advance and depend on the best available view of future loads. A survey by Rashid et al. (2024) reveals that research on EV demand forecasting has mainly pursued two related goals: higher accuracy (minimising error against realised outcomes), and higher realism (capturing the inherently uncertain and variable nature of real-world demand). Different methods have been employed to achieve these aims.

2.2.1 Classical statistical models

Marzbnai et al. (2023) demonstrate how early work adapted MLR and Seasonal Autoregressive Integrated Moving Average (SARIMA) models for EV demand forecasting. These models are transparent and data-light, making them accessible for local authorities (Biswal et al., 2024). However, their underlying assumptions, such as linearity and stationarity, limit their ability to represent non-linear and drifting dynamics, as EV demand shifts with adoption rates, tariffs, weather, seasonality, and local habits (Zhou et al., 2025). Consequently, while classical models offer transparency and a low data burden, their rigidity limits their applicability in dynamic EV

contexts where relationships evolve over time. This limitation has driven the adoption of more sophisticated models.

2.2.2 Machine-learning (ML) models

Shern et al. (2024) report improved performance from support vector regression (SVR) and ensemble methods, such as extreme gradient boosting (XGBoost). Using richer features, such as calendar effects and weather, tends to improve accuracy (Rashid et al., 2024; Shams et al., 2021), because ML models can capture interactions and non-linear effects that simpler statistical forms miss. Comparative studies also show sizeable error reductions over statistical baselines, providing a stronger insight for operational planning (Aldossary et al., 2024; Shahriar et al., 2021) However, these gains come with trade-offs: many ML models are data-hungry and comparatively opaque, complicating validation and audit (Halevy et al., 2009; Raji et al., 2020; Rudin, 2019). Feature sets relying on location-specific or platform-specific data can also encode coverage gaps; under-representation of lower-income or rural users in training data risks forecast errors across communities (Buolamwini & Gebru, 2018; Corbett-Davies et al., 2023; Mehrabi et al., 2022). The shift toward ML reflects a broader tendency in energy systems to favour data-driven adaptability, though tempered by concerns about explainability and data equity (Akter et al., 2021; Ejiyi et al., 2025; Mocanu et al., 2016). Some scholars have also explored more sophisticated methods, such as learning models.

2.2.3 Deep learning (DL) Models

DL has been used to capture richer temporal structure in EV demand. Agga et al. (2022) and Vishnu et al. (2023) show that hybrid convolutional and long short-term memory networks (CNN-LSTMs) can learn both short local patterns and longer dependencies, aiding in situations where daily cycles and event-driven spikes overlap. Koohfar et al. (2023) show that transformer-based models can outperform ARIMA, SARIMA, RNN, and LSTM baselines, making them effective for longer horizons. For EV demand, attention mechanisms help represent long-range temporal effects, such as holiday calendars and adoption trends, alongside short spikes, while graph-based layers can encode spatial coupling across stations and feeder areas (Zhang et al., 2023).

The review indicates that DL and ML models improve fit on complex patterns. However, most models remain deterministic, outputting a single value without providing confidence information (Cao et al., 2024), which limits risk management because operators cannot assess

the fragility of a schedule if demand shifts (Zhou et al., 2016). This limitation necessitates the incorporation of methods that account for uncertainty.

2.2.4 Probabilistic forecasting

Probabilistic methods address this missing confidence signal. Gneiting & Katzfuss (2014), refer to this method as forecasting approaches that return a predictive distribution or intervals, rather than a single point estimate, making uncertainty explicit and usable in decision-making. Foundational work by Hong & Fan (2016) and Pinson (2013) in energy forecasting, sets out the case for decision-oriented uncertainty. Quantile regression produces prediction intervals that bound likely outcomes (Bracale et al., 2020; Cao et al., 2024; Yin et al., 2023), enabling operators to translate bounds into headroom or curtailment rules at sites. Quantile objectives can also be paired with boosting models, such as XGBoost, to deliver interval forecasts, thereby considering uncertainty (Kangalli Uyar et al., 2025). The broader trend is a shift from deterministic estimates to decision-grade distributions that can be consumed by downstream optimisation.

Taken together, these developments in forecasting set the stage for the operational question that follows: how to act on such predictions within operational and grid constraints?

2.3 Conventional operational practice: uncontrolled and rule-based charging

In an uncontrolled system, vehicles draw power as soon as they plug in. Studies indicate that this behaviour amplifies peaks, heightens demand-charge exposure, and stresses local assets, with issues like voltage excursions and thermal loading, most likely around commuter windows (Chang et al., 2021; Clement-Nyns et al., 2010; Richardson et al., 2012). The practical implication is that even moderate uptake can create disproportionate costs and reliability risks during periods of peak demand. Operators consequently face a choice: overbuild capacity or accept higher risk (Muratori, 2018).

Rule-based schemes attempt to mitigate this by employing simple heuristics, such as charging only during fixed off-peak hours or capping site power at a predetermined threshold. They offer ease of implementation and explainability, and can shift energy into cheaper periods (Das et al., 2020). Yet their rigidity is a weakness. When many users follow the same window, demand peaks emerge, eroding savings and pushing sites against constraints; fixed caps also fail to react to live conditions (Chang et al., 2021; Uralde et al., 2024). These limitations motivate more anticipative approaches that compute schedules against an explicit view of near-term demand.

Thus, the field is shifting towards smart charging, which acts on forecasts rather than reacting to outcomes.

2.4 EV Smart charging

EV smart charging is the adaptive scheduling and control of charging to align with grid constraints, user demand, and costs (Brinkel et al., 2024). It supports demand-side flexibility and increases renewable utilisation by shifting charging to align with low-carbon generation windows (Manousakis et al., 2023). In line with this, the UK's "EV Smart Charging Action Plan" outlines regulatory support for systems that optimise charging based on grid signals (GOV.UK, 2023). Smart charging is also seen as a way to aid grid stability through vehicle-to-grid technology (Mwasilu et al., 2014).

2.5 Scheduling formulations for EV smart charging

One way of framing the charging schedule problem is as a constraint-based optimisation that represents the task as a linear programming (LP) or mixed-integer linear programming (MILP) problem that finds the least-cost plan subject to explicit feasibility limits, such as meeting total demand, without exceeding charger capacity (Ortega-Vazquez, 2014; Yang et al., 2022). In practice, these models are often embedded on a rolling horizon as model predictive control (MPC), allowing the schedule to be re-optimised as new forecasts arrive; studies report gains for load balancing and time-of-use prices (Li et al., 2024). Hybrids that feed short-term machine-learning forecasts into the MPC loop can further improve responsiveness and cost efficiency (Ivanovich Vatin & Madhavi, 2024). However, performance hinges on the quality and frequency of forecast refresh (Mayne, 2000). Some studies have also used reinforcement learning for scheduling problems, treating it as sequential decision-making that learns and reacts to stochastic arrivals and non-linear costs (Vázquez-Canteli & Nagy, 2019). This can be helpful when dynamics are complex or partially understood. However, it can be computationally intensive, and performance depends on reward design, safe exploration, and the realism of training data (Dulac-Arnold et al., 2019). Whichever method is chosen, the optimisation plan inherits whatever the forecast says will happen next.

2.6 Forecast-informed Schedule Optimisation

Evidence consistently links better forecasts to better schedules. Aldossary et al. (2024) demonstrated that improved demand forecasts, lowered peaks, and improved station operations. Similarly, a case study at Politecnico di Milano showed that higher-accuracy

forecasts of EV demand reduce operating costs in a microgrid by enabling more precise MILP optimisation (Osama, 2024). Xu et al. (2025) achieved a 15% reduction in peak load and a 23% increase in clean energy usage by using a ML-enhanced dispatch. The core mechanism is that the optimiser plans against the forecast profile. However, a critical challenge exists: if peaks or totals are misestimated, charging is pushed into the wrong hours or volumes, thereby increasing costs. Therefore, since plans inherit forecast errors, deterministic set-ups are fragile once conditions shift (Grimes et al., 2014; Soroudi & Keane, 2014). Consequently, forecast design is not merely a modelling choice but an operational lever: errors propagate into cost, service quality, and constraint violations. Hence, reinforces the need for probabilistic modelling that quantifies uncertainty.

2.7 Scheduling under uncertainty

Uncertainty-aware optimisation incorporates probabilistic forecasts directly. One approach is stochastic programming, which aims to find policies that perform well on average across many plausible scenarios by solving the problem with multiple sampled futures and weighting their outcomes (Bertsimas et al., 2010). Another is robust optimisation, which seeks solutions that remain feasible under adverse conditions by enforcing constraints across an uncertainty set, thereby building deliberate slack into schedules (Soroudi & Keane, 2014). These methods can be computationally demanding because scenario trees or worst-case constraints enlarge the optimisation problem (Faridimehr et al., 2019), but they are designed for the volatility seen in practice. On the forecasting side, techniques such as quantile regression and Monte Carlo dropout enable models to be trained to provide prediction intervals that can be passed to the optimiser, allowing constraints and objectives to reference ranges rather than single points (Cai et al., 2022; Vergara et al., 2019). Empirical results suggest higher charging success rates, lower peaks, and better cost control when such uncertainty estimates drive the optimisation (Zheng et al., 2025). Together, these approaches reframe scheduling as a risk-aware decision made under distributions rather than a deterministic optimisation against a single trajectory.

2.8 Synthesis and Research Gaps

Despite policy momentum and technical advances, three interlinked gaps remain. First, socio-technical analysis of user behaviour and equity is often disconnected from forecasting and optimisation workflows. Second, many studies still rely on single-point forecasts, overlooking the uncertainty associated with them. Third, optimisation frameworks are commonly tested under perfect foresight rather than real-world volatility.

Given the Committee of Public Accounts' evidence on uneven access and the UK policy push for smart, price-responsive charging, an equity-sensitive and uncertainty-aware framework is necessary. Behavioural variability across geography and income groups makes point forecasts brittle; probabilistic forecasting and robust optimisation are the technical levers that translate these socio-behavioural realities into reliable operations. Informed by this analysis, this dissertation confronts these gaps by developing and testing a technical framework that embeds uncertainty-aware forecasting within an optimisation model, grounded in a real-world socio-technical case study (PKC). By building this integrated workflow, it addresses key questions of forecasting accuracy and optimisation gains and offers methodological advances with practical insights for operators. Chapter 3 now sets out the methodology.

Chapter 3: Methodology

The preceding chapter established the theoretical context for this study and identified gaps in the existing literature. This chapter addresses these gaps by detailing the systematic methodology designed to build and evaluate the end-to-end, uncertainty-aware optimisation framework. It outlines the research design, data sources, and the sequential pipeline of data preparation, forecasting, and optimisation that forms the empirical core of this dissertation.

3.1 Research Design and Framework

The research is designed as a quantitative, data-driven case study of the public EV charging network in PKC, UK. This approach ensures the findings are empirically grounded in real-world usage patterns, allowing for an analysis that is both technically robust and capable of revealing the socio-economic impacts of different charging strategies.

The methodology is structured around a sequential, four-stage pipeline, hereafter referred to as the Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF), as illustrated in Figure 2. The process begins with a socio-technical analysis, followed by data preparation, the development of an uncertainty-aware forecast, and finally, a simulation where the forecast informs a cost-optimising linear programming model. The outputs are then compared against baseline strategies to evaluate performance.

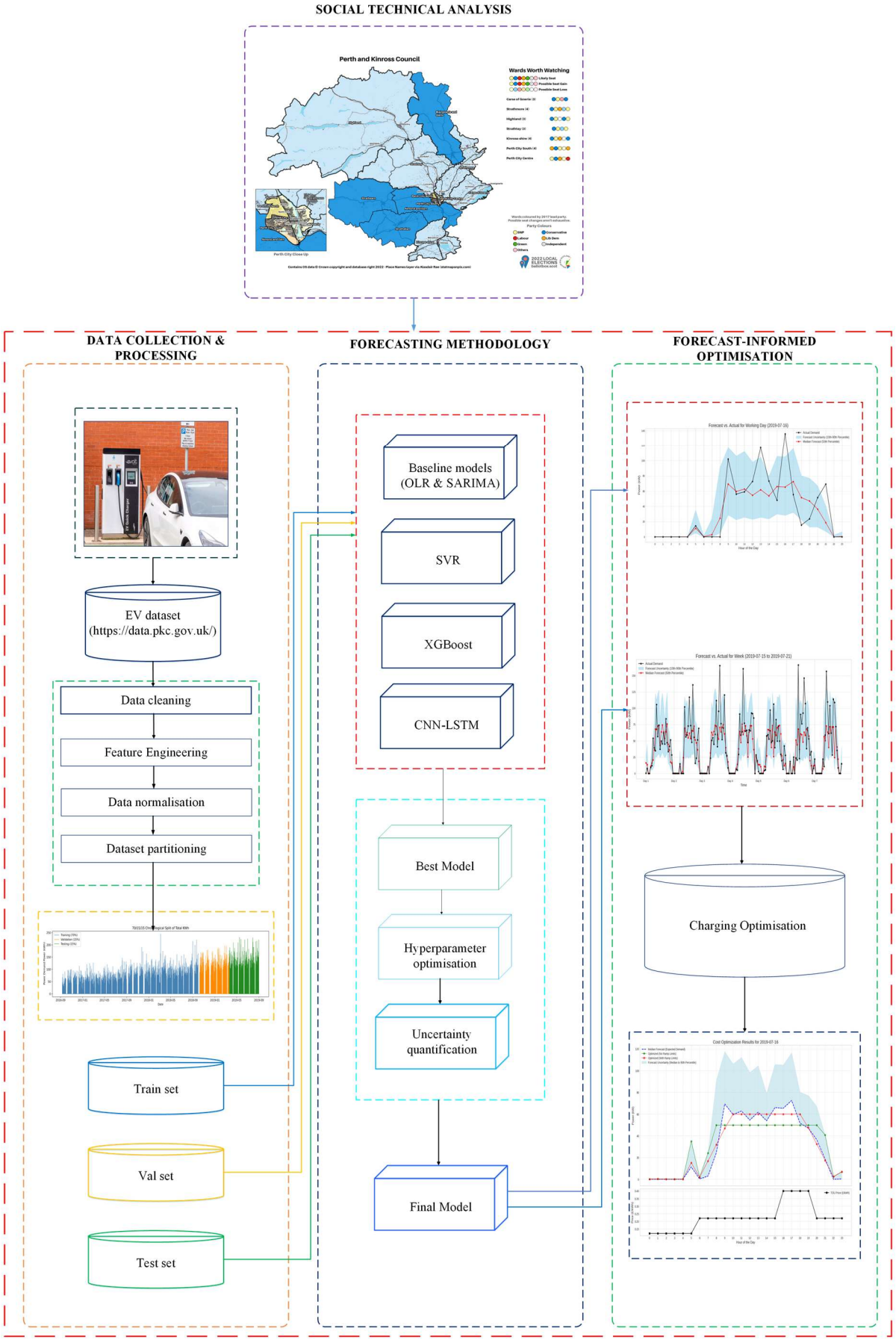


Figure 2: Overview of the four-stage ST-UOF.

3.2 Data Source

To conduct this analysis, two datasets were sourced from PKC's public data portal. The dataset for forecasting consists of 55,878 anonymised charging records across PKC's EV charging stations between September 2016 and August 2019. This historical data provides the real-world usage patterns necessary to train and validate the machine learning models. The work aims to forecast the hourly energy demand in the network, rather than being specific to any charging station. It is essential to note that the data includes network-level information. A second dataset contains the geographic and technical specifications for all 45 public charging points active as of March 2023. This is crucial for the socio-technical analysis of infrastructure distribution and accessibility. The specific attributes for both datasets are detailed in Appendix A.

3.3 Exploratory Analysis: The Socio-Technical Context of PKC

This research adopts a "socio-technical first" approach, using the current infrastructure data to establish the real-world context before proceeding to the technical modelling. This exploratory analysis focused on four key areas:

- **Geographic Equity Analysis:** Plotting charger locations and speeds to visually assess potential "charging deserts" and the distribution of high-speed infrastructure.
- **Economic Accessibility Analysis:** Analysing the distribution of "Free" versus "Paid" chargers to understand the network's economic model and potential cost barriers.
- **Technology Equity and Futureproofing:** Assessing the prevalence of different connector standards (CCS, CHAdeMO, Type 2) to analyse the network's long-term viability and compatibility for owners of modern EVs.
- **Cost vs Convenience Trade-off:** I compare the uncontrolled charging curve with the forecast-optimised schedule under illustrative Time-of-Use prices to visually illustrate how deferring load into cheaper hours cuts costs but may inconvenience drivers.

By establishing this socio-technical baseline, the research ensures that subsequent quantitative efforts are grounded in the understanding of the existing system's strengths, weaknesses, and equity implications. The next step was to begin processing the dataset for model training.

3.4 Data Pre-processing and Feature Engineering

The data pre-processing and feature engineering pipeline was implemented in Python to transform the raw datasets into a clean format suitable for machine learning.

3.4.1 Data Cleaning and Aggregation

The initial data cleaning involved standardising column names, removing records with missing critical information such as timestamps or coordinates, and filtering out invalid sessions with zero or negative energy delivered from the historical usage dataset.

3.4.2 Time Series Aggregation

Following Vishnu et al. (2023), individual charging events were resampled into hourly intervals. For each hour, the total energy delivered was summed to create an aggregate hourly demand signal in kWh, which was established as the target variable for the forecasting models. Figure 3 shows a plot of the hourly average demand. From the plot, charging demand at PKC is higher during active hours of the day, with the highest values recorded at around 12:00 p.m. Also, the lowest demand is visible from 12:00 a.m. to 4:00 a.m. This signals an uncontrolled charging setting, which is later assumed as the current baseline for the charging strategies.

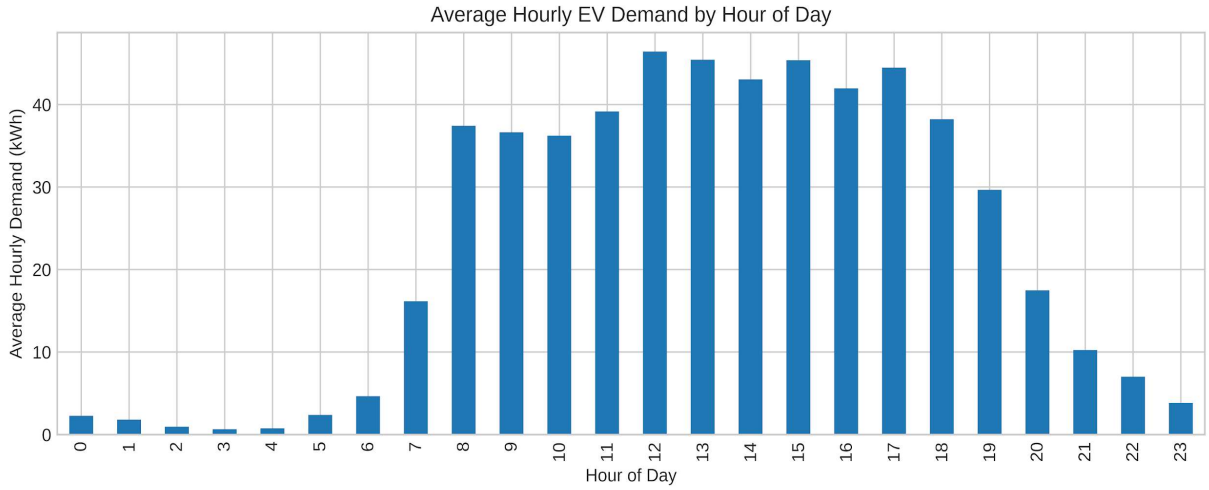


Figure 3: Average hourly EV demand by hour of the day.

3.4.3 Feature Engineering

To provide the models with signals known to influence charging demand, several features were engineered (Elahe et al., 2022). These included time-based features (e.g., hour of the day), rolling-window features (e.g., previous 24-hour average demand), and categorical encoding for seasonal and day-of-the-week patterns.

3.4.4 Feature Selection and Correlation Analysis

To reduce model complexity and mitigate multicollinearity, feature selection was performed using Pearson correlation coefficients, visualised in a heatmap (Figure 4) (Bouchlaghem et al.,

2022). Redundant features with very high correlation (e.g., >0.99) were dropped. The final set of features used for model training is provided in Appendix A, Table 7.

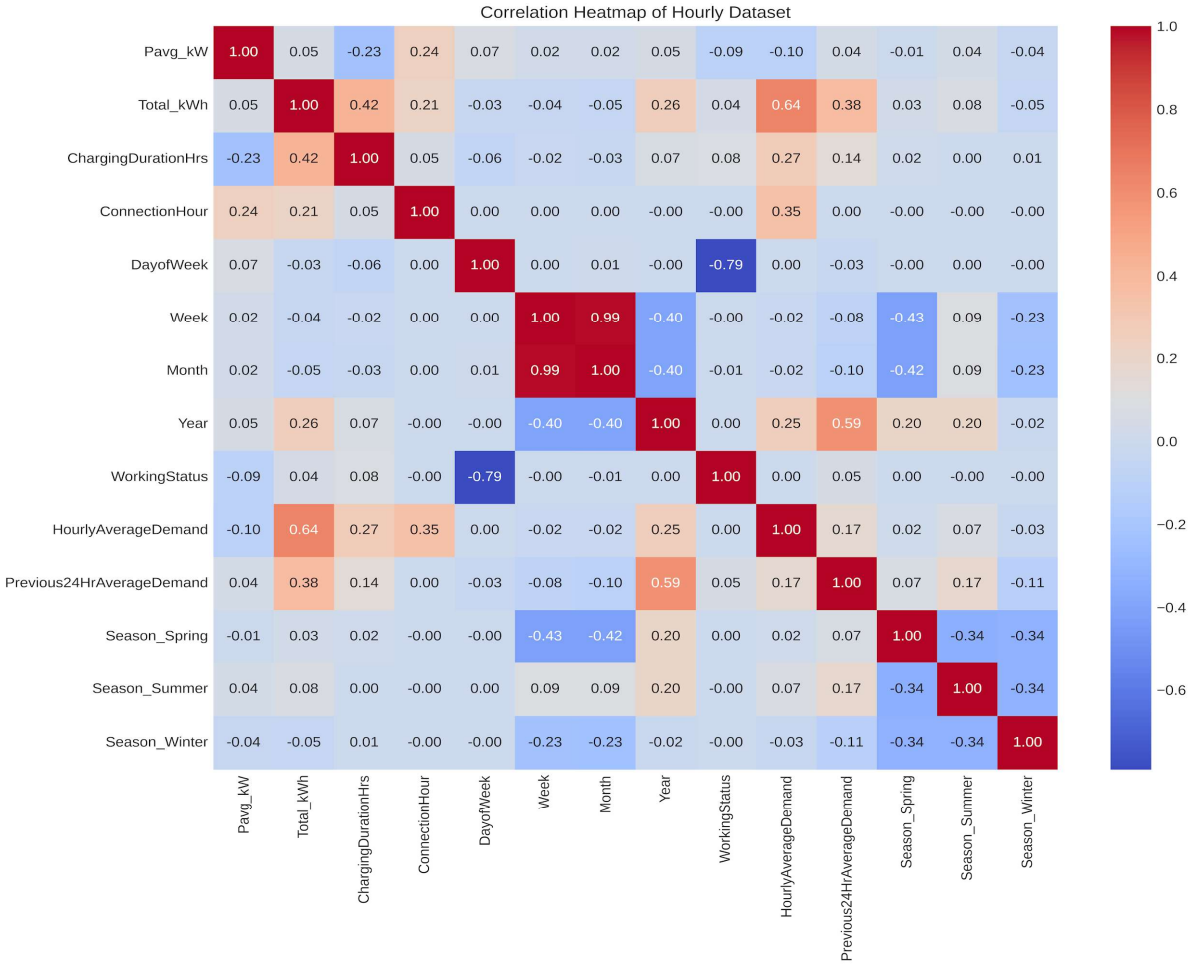


Figure 4: Correlation matrix of all model features.

3.4.5 Outlier Detection

To handle extreme values, outlier detection was performed on the session-level energy data using the Interquartile Range (IQR) method (Mahajan et al., 2020; Rousseeuw & Leroy, 1987). IQR is the difference between the third and first quartiles. This method, which has proven effective in similar studies, is calculated using Equations (1) to (3) (Vishnu et al., 2023). Any charging session with a `Total_kWh` value falling outside the calculated IQR bounds was removed from the dataset prior to model training, as illustrated in Figures 5 and 6. The quantitative details of this process are available in Appendix A, Table 8.

$$IQR = Q3 - Q1 \tag{1}$$

$$Lower\ Bound: Ll = Q1 - 1.5 \times IQR \tag{2}$$

$$Upper\ Bound: Lu = Q3 + 1.5 \times IQR \tag{3}$$

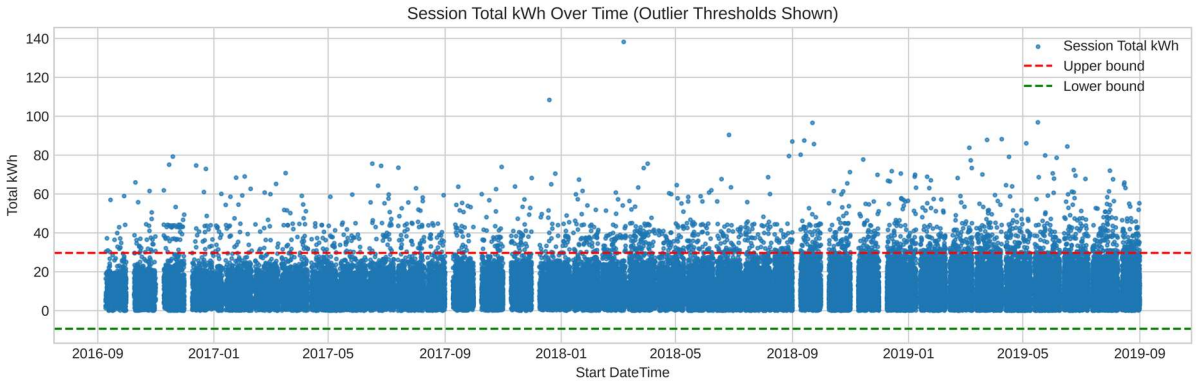


Figure 5: Charging session data before outlier removal.

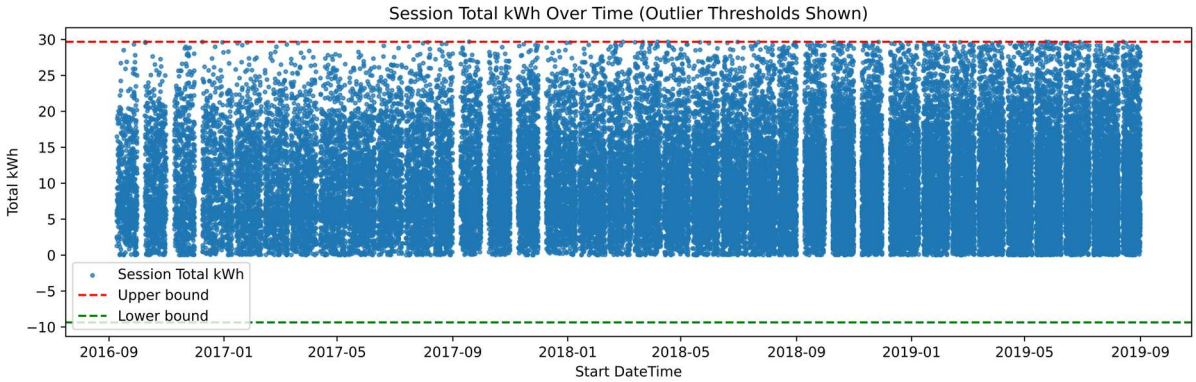


Figure 6: Charging session data after outlier removal.

3.5 Forecasting Model Development and Evaluation

With a prepared dataset, the research proceeded to develop and evaluate a range of forecasting models to address RQ2, starting with simple baselines (MLR via Ordinary Least Squares (OLS) and SARIMA) and progressively moving to more complex models (SVR, XGBoost, CNN-LSTM), culminating in an uncertainty-aware forecasting system.

3.5.1 Data Splitting and Scaling

The hourly time series data was split chronologically into training (70%), validation (15%), and testing (15%) sets to prevent data leakage (Figure 7) (Bergmeir & Benítez, 2012). A chronological split is necessary to prevent the model from inadvertently being trained on information from the future.

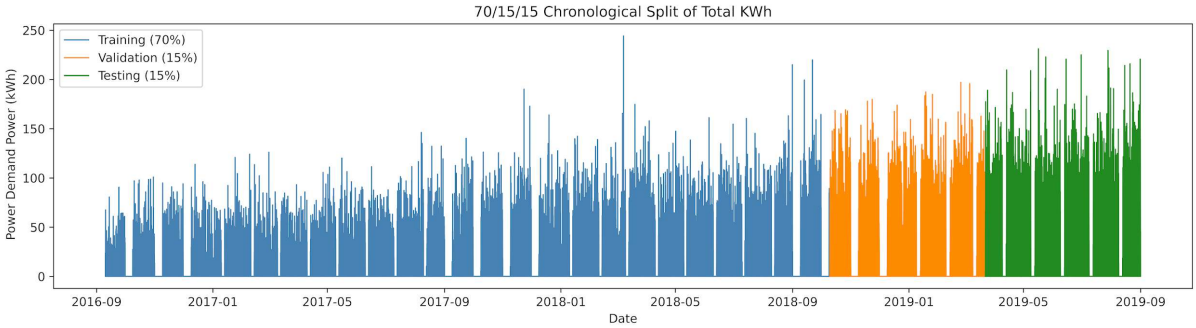


Figure 7: Chronological split of the dataset.

Following this, all numerical features were scaled using Min-Max Normalisation, a technique that transforms values to a fixed range of [0, 1]. This method was chosen because studies show it is effective for training ML and DL models (de Amorim et al., 2022; Van Kriekinge et al., 2021). The normalised value is calculated as shown in Equation (4) (Elahe et al., 2022).

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4}$$

Where x is the original data point, x_{norm} is the normalised data point, and x_{max} and x_{min} are the maximum and minimum values of the feature, respectively. The scaler was fitted only on the training data to prevent any information leakage from the validation or test sets into the training process.

3.5.2 Modelling Approaches

The next step was to develop and train the selected models. The models used in this study, along with their underlying principles, are detailed below.

3.5.2.1 Multiple Linear Regression (Baseline)

The initial baseline model is an MLR. This model assumes a linear relationship between the input features and the target variable. It provides a performance benchmark against which more complex models can be evaluated. The mathematical representation is presented in Equation (5) (Shalev-Shwartz & Ben-David, 2013).

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon \tag{5}$$

Where Y is the predicted demand, $X_1, \ldots, X_n$ are the input features, $\beta_1, \ldots, \beta_n$ are the coefficients learned by the model, β_0 is the intercept, and ϵ is the error term.

3.5.2.2 Seasonal Autoregressive Integrated Moving Average (SARIMA)

The SARIMA model was also selected as a statistical baseline. Unlike the ML models, which rely on exogenous features, it models the target variable based solely on its past values and past error terms (Alharbi & Csala, 2022). Its ability to model trend, seasonality, and autoregressive components makes it a valuable benchmark for time series data (Akshay et al., 2024). Its general form can be expressed as shown in Equation (6) (Manigandan et al., 2021).

$$\varphi_p(G)\varphi_P(G^s)\,(1-G)^d\,(1-G^s)^D\,X_t \;=\; \gamma_q(G)\,\omega_Q(G^s)\,e_t \tag{6}$$

Where:

- X_t is the observed value at time t.
- G is the backshift operator (i.e., $GX_t = X_{t-1}$).
- (p, d, q) are the non-seasonal orders for Autoregressive (AR), differencing (I), and Moving Average (MA) components.
- $(P, D, Q)_s$ are the seasonal orders for the seasonal AR, differencing, and MA components, with s being the seasonal period (s = 24 for daily seasonality in hourly data).
- $\varphi_p(G)$ and $\gamma_q(G)$ are the non-seasonal AR and MA characteristic polynomials
- $\varphi_P(G^s)$ and $\omega_Q(G^s)$ are the seasonal AR and MA characteristic polynomials.
- e_t is the prediction error term at time t.

3.5.2.3 Support Vector Regression (SVR)

SVR is a supervised learning ML model; it was chosen for its ability to capture complex, non-linear relationships common in volatile energy demand data without making strong assumptions about the data's distribution (Pai & Hong, 2005). Unlike linear models, SVR uses a non-linear kernel to map inputs into a higher-dimensional space. This "kernel trick" allows it to model nuanced feature interactions, such as those between time of day and seasonality (Gambella et al., 2021; Hsu et al., 2003). The model operates by identifying a function that accurately fits the training data within a specified error margin. Its use in short-term load forecasting is well-documented, and the underlying equations are shown from (7) to (11) (Vishnu et al., 2023).

$$K(x, x') = \big(\phi(x), \phi(x')\big) \tag{7}$$

Similar to Shahriar et al. (2021), Radial Basis Function (RBF) kernel was used. The optimisation problem for SVR is to find the flattest function by minimising the norm of the weight vector, *w*, while tolerating some error. This is formulated as:

$$Minimise: \frac{1}{2} \| \omega \|^2 + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \quad (8)$$

Subject to:

$$y_i - f(x_i) \leq \epsilon + \xi_i \quad (9)$$

$$f(x_i) - y_i \leq \epsilon + \xi_i^* \quad (10)$$

$$\xi_i, \xi_i^* \geq 0 \quad (11)$$

Where *C* is the regularisation parameter that trades off the flatness of *f(x)* and the amount up to which deviations larger than ϵ are tolerated, and ξ_i ξ_i^* are slack variables representing the magnitude of this deviation for each data point.

3.5.2.4 XGBoost (Ensemble Method)

XGBoost, a gradient boosting ensemble method, was selected for its ability to handle complex feature interactions while preventing overfitting through built-in regularisation (Chen & Guestrin, 2016; Shahriar et al., 2021). The algorithm sequentially builds an ensemble of decision trees, summing their outputs to make a final prediction. To validate its selection, a comparative evaluation was conducted, comparing it to other ensemble learning methods (Random Forest & LightGBM). The results are detailed in Appendix B, Table 9, and the final parameters used to train the model are presented in Table 10. The mathematical formulation is given in Equations (12) and (13) (Shern et al., 2024).

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \quad (12)$$

Where f_k is the prediction function of the ***k*-th** tree, the model is trained iteratively, at each step ***t*,** a new tree f_t is added to minimise the objective function $L^{(t)}$, which combines a loss function and a regularisation term that penalises the complexity of the model:

$$L^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_i(x_i)) + \Omega(f_i) \quad (13)$$

Here:

- $l(\cdot)$ is a differentiable convex loss function (e.g., Mean Squared Error for regression).
- $\hat{y}_i^{(t-1)}$ is the prediction from the ensemble before adding f_t.
- $\Omega(f_i)$ is a regularisation term penalising the complexity of the new tree f_t, typically based on the number of leaves and the L_2 norm of the leaf weights

3.5.2.5 Hybrid Deep Learning Model: CNN-LSTM

A hybrid deep-learning CNN-LSTM model was also explored for its proven ability to capture spatial and temporal dependencies in energy demand, such as local daily patterns and longer weekly trends (Agga et al., 2022). It combines the feature extraction of Convolutional Neural Networks (CNNs) with the sequence modelling of Long Short-Term Memory (LSTM) networks – an approach with significant promise in energy forecasting (Lai et al., 2018).

The hybrid model architecture processes an input window of 24 hours with 14 features. The pipeline consists of four main stages, as shown in the complete architecture diagram in Figure 10:

1. **CNN Feature Extraction**: The input data first passes through a 1D Convolutional layer with 64 filters and a kernel size of 3, using a Rectified Linear Unit (ReLU) activation function. This process of using filters to extract local features is visualised in Figure 8. Batch Normalisation and a Dropout layer with a rate of 0.3 follow this layer.
2. **Sequence Learning**: The extracted features are then fed into a sequence of two LSTM layers, each with 50 units and a tanh activation function. The internal structure of an LSTM cell is shown in Figure 9. A Dropout layer with a rate of 0.4 is applied after each LSTM layer to prevent overfitting.
3. **Dense Layer**: The output from the second LSTM layer is passed through a dense layer with 25 units and a ReLU activation function.
4. **Final Forecast**: The final output layer consists of a single unit with a linear activation function to produce the single-point forecast.

The complete model architecture is shown in Figure 10 below, and the underlying mathematical equations for the LSTM gates are detailed in Equations (14) to (18) below (Shams et al., 2021)

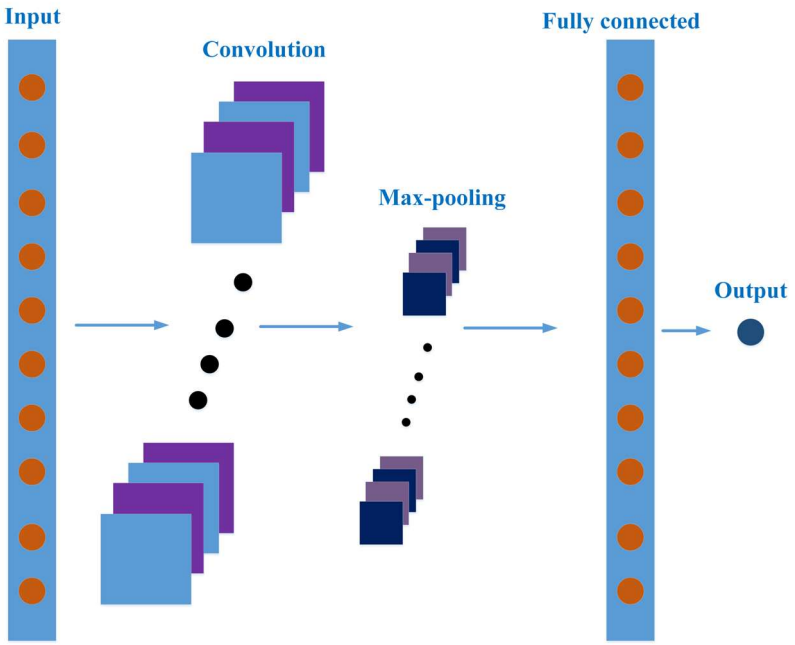


Figure 8: A single-layer CNN architecture, adapted from Agga et al. (2022).

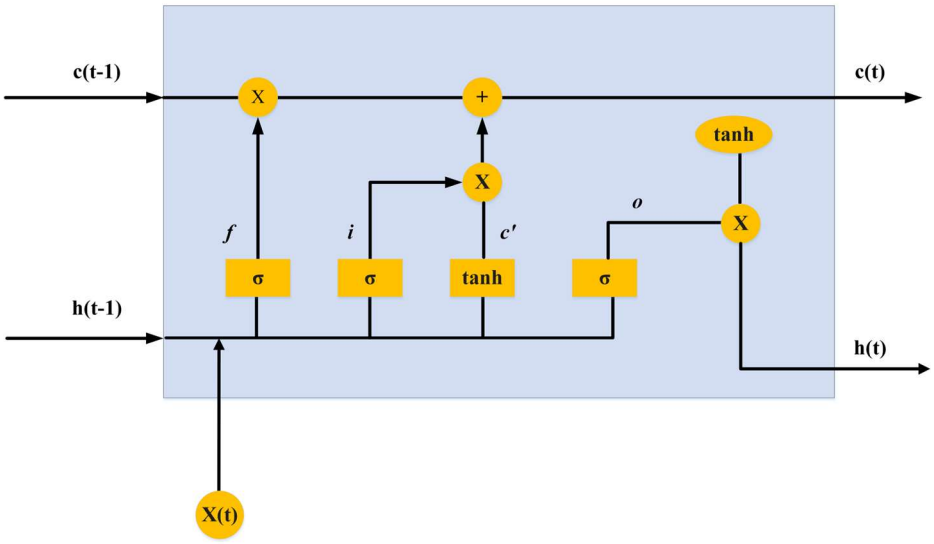


Figure 9: The internal architecture of an LSTM cell, adapted from Agga et al. (2022).

- Input Gate: $i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$ (14)
- Forget Gate: $f_t = \sigma(W_{xj}x_t + W_{hf}h_{t-1} + b_f)$ (15)
- Output Gate: $o_t = \sigma(W_{x0}x_t + W_{h0}h_{t-1} + b_0)$ (16)
- Cell State Update: $c_t = f_t \odot c_{t-1} + i_t \odot tan\ h\ (W_{xc}x_t + W_{hc}h_{t-1} + b_c)$ (17)
- Hidden State Update: $h_t = o_t \odot tan\ h\ (c_t)$ (18)

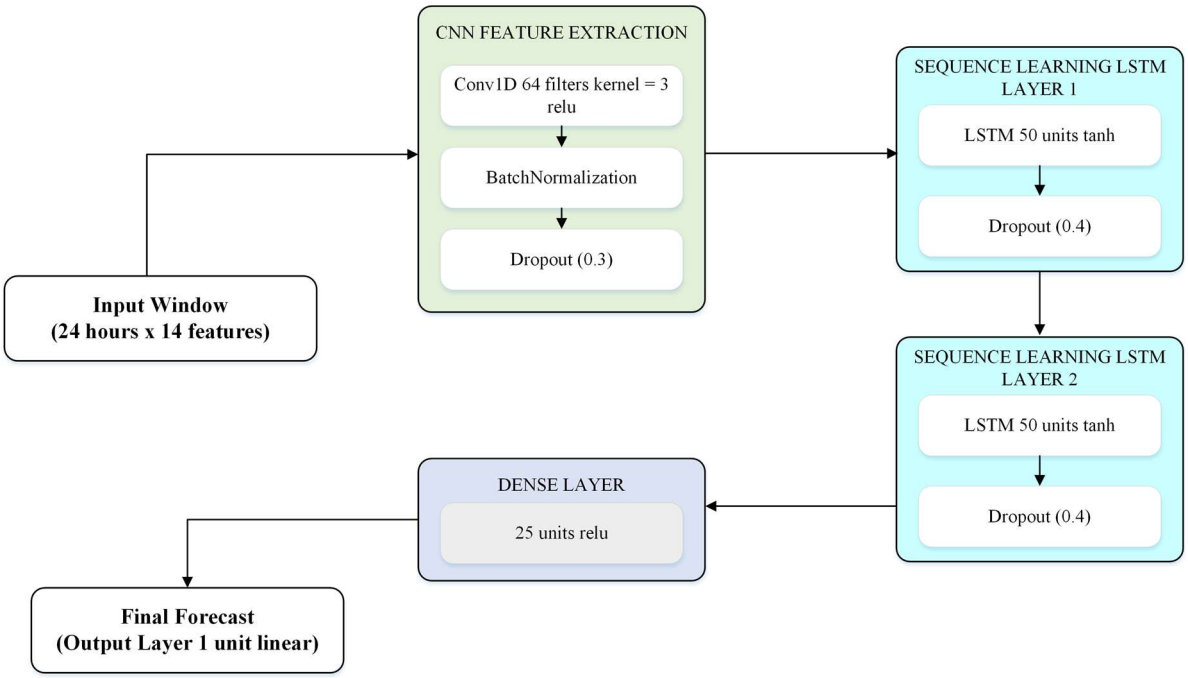


Figure 10: Hybrid CNN-LSTM Model Architecture

3.5.3 Performance Evaluation Metrics

The performance of the models was quantitatively compared on the unseen test set using three standard regression metrics. This approach follows Ran et al. (2025). The metrics are defined in Equations (19)–(21).

- **Mean Absolute Error (MAE):** Measures the average magnitude of the errors in a set of predictions, without considering their direction.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{19}$$

- **Root Mean Squared Error (RMSE):** This is the square root of the average of squared differences between prediction and actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{20}$$

- **R-squared (R^2):** This indicates the goodness of fit of a set of predictions to the actual values. It represents the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (21)$$

Where **n** is the number of samples, $\boldsymbol{y_i}$ is the actual value, $\boldsymbol{\hat{y}_i}$ is the predicted value, and $\boldsymbol{\bar{y}}$ is the mean of the actual values.

3.5.4 Uncertainty Quantification with Quantile Regression

Recognising the limitations of single-point forecasts, the methodology was extended to quantify uncertainty. Quantile regression was applied to the best-performing model (XGBoost) to forecast the 10th, 50th, and 90th percentiles of the demand distribution. This approach provides a probability forecasting interval, offering a range of likely outcomes rather than a single estimate, which is crucial for real-world decision-making (Cai et al., 2022; Siddique et al., 2022), and also a key element in my methodological framework.

3.6 Forecast-Informed Optimisation Methodology

The second research phase addresses RQ1 by using the forecasts to inform an operational optimisation framework that computes a least-cost charging schedule. Linear Programming (LP) was adopted for its ability to efficiently model system dynamics like capacity and ramp-rate restrictions as linear relationships (Bertsimas et al., 2010; Ortega-Vazquez, 2014). While more advanced methods exist, their complexity exceeded this study's scope. The LP model was implemented in Python using `PuLP`, a library that allows for concise declaration of decision variables, objectives, and constraints.

3.6.1 Formulation of Economic Parameters

The dataset lacked historical electricity prices, so a representative Time-of-Use (TOU) tariff was formulated to create a realistic economic structure for the optimisation model. This tariff divides the day into three tiers – off-peak, standard, and peak – based on typical UK commercial rates (Ingrams, 2025). Furthermore, a £12/kW demand charge was incorporated to penalise the single highest power peak recorded during the period, reflecting a critical component of commercial electricity billing. The impact of varying this charge on the final savings is explored in the sensitivity analysis (Appendix C). These parameters ensure the optimisation problem addresses the real-world trade-offs faced by a charge point operator.

3.6.2 Mathematical Formulation

The optimisation problem was formulated as a Linear Program designed to minimise the total daily electricity cost, which includes both the Time-of-Use (TOU) energy price and a peak demand charge. The model schedules the hourly charging power subject to several key operational constraints: it must meet the total energy demand forecast by the median prediction, it cannot exceed the maximum power level set by the 90th percentile forecast, and it must adhere to maximum ramp rates. The detailed mathematical formulation, including all equations and parameters, is provided in Appendix F.

3.6.3 Simulation of Charging Strategies

The final stage of the methodology involves evaluating the performance of the optimisation framework. A comparative analysis was conducted by simulating three distinct charging strategies over daily, weekly, and monthly horizons:

- **Uncontrolled Charging (Baseline):** This strategy represents the assumed current, unmanaged condition of the stations. It uses the actual historical demand from the test set as a proxy for convenience-driven, uncoordinated charging. This assumption is supported by the network's demand profile (Figure 3), which clearly shows that charging peaks during convenient but expensive daytime hours – a hallmark of unmanaged usage. It therefore provides a realistic baseline for cost and performance against which the other, more innovative strategies can be measured.
- **Rule-Based Charging:** This strategy mimics a simple, non-predictive smart charging approach where a static rule permits charging only during pre-defined "solar-rich" hours (11 a.m. to 3 p.m.). The total daily energy demand is spread evenly across this fixed window. While simple, this method risks creating a new, artificial power peak, the adverse financial consequences of which are demonstrated in the sensitivity analysis in Appendix C.
- **Forecast-Informed Optimisation:** This is the strategy developed in this research and represents the final output of the Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF). It uses the complete optimisation model described in Section 3.6 to generate a cost-optimal schedule.

The primary output of this simulation is a comparison of total operational costs, which is used to quantify the savings achieved by the rule-based and forecast-informed strategies relative to the uncontrolled baseline.

3.7 Implementation and Ethical Considerations

The analysis was conducted in Python 3 using the Google Collaboratory browser interface. It relied on a suite of open-source libraries, including `Pandas`, `NumPy`, `Scikit-learn`, `XGBoost`, `TensorFlow/Keras`, `Optuna`, `PuLP`, `Matplotlib`, and `Seaborn`. The complete code and environment details are provided in Appendix E.

This research was conducted in accordance with ethical standards. The primary dataset, sourced from Perth and Kinross Council's public portal, was fully anonymised and contained no personally identifiable information, complying with UK Data Service guidelines (UK Data Service, 2025). Ethical approval was obtained from the University of Warwick's Centre for Interdisciplinary Methodologies prior to analysis. The use of large language models for code debugging is acknowledged and has been employed, as stated in the Academic Integrity Declaration (Page 9).

Chapter 4: Results

The previous chapter described the data-gathering methodology and explained the theoretical considerations underlying this research. Based on the objectives highlighted in Chapter 1, this chapter presents the empirical findings of the research, organised sequentially to mirror the methodological pipeline. It begins with an exploratory socio-technical analysis of the Perth and Kinross (PKC) public charging infrastructure to establish the network's real-world context. The chapter then details the evaluation of the forecasting model to identify the most accurate model, before presenting the results of incorporating uncertainty into the model. Finally, it presents the performance of the forecast-informed optimisation and quantifies the cost savings achieved across different time horizons.

4.1 Exploratory Analysis: The Socio-Technical Landscape of PKC's Network

The initial stage of my research examines the existing public EV charging infrastructure in PKC, establishing a baseline of network layout, strategic priorities and equity gaps. It also introduces a visualisation of the forecast-informed optimisation, illustrating the trade-off between cost savings and user convenience.

4.1.1 Geographic and Usage-Based Equity

Figure 11 maps all Rapid and Fast/Standard chargers, revealing a dense cluster in central Perth, in contrast to sparse provision in outskirt areas. While this provides good coverage for key towns, it also highlights a comparative lack of infrastructure in the more remote parts of the region. Figure 12 ranks the top 15 sites by cumulative energy delivered (2016–2019), showing that Park & Ride locations (e.g. Broxden, Kinross) dominate network utilisation and experience the most significant operational pressure.

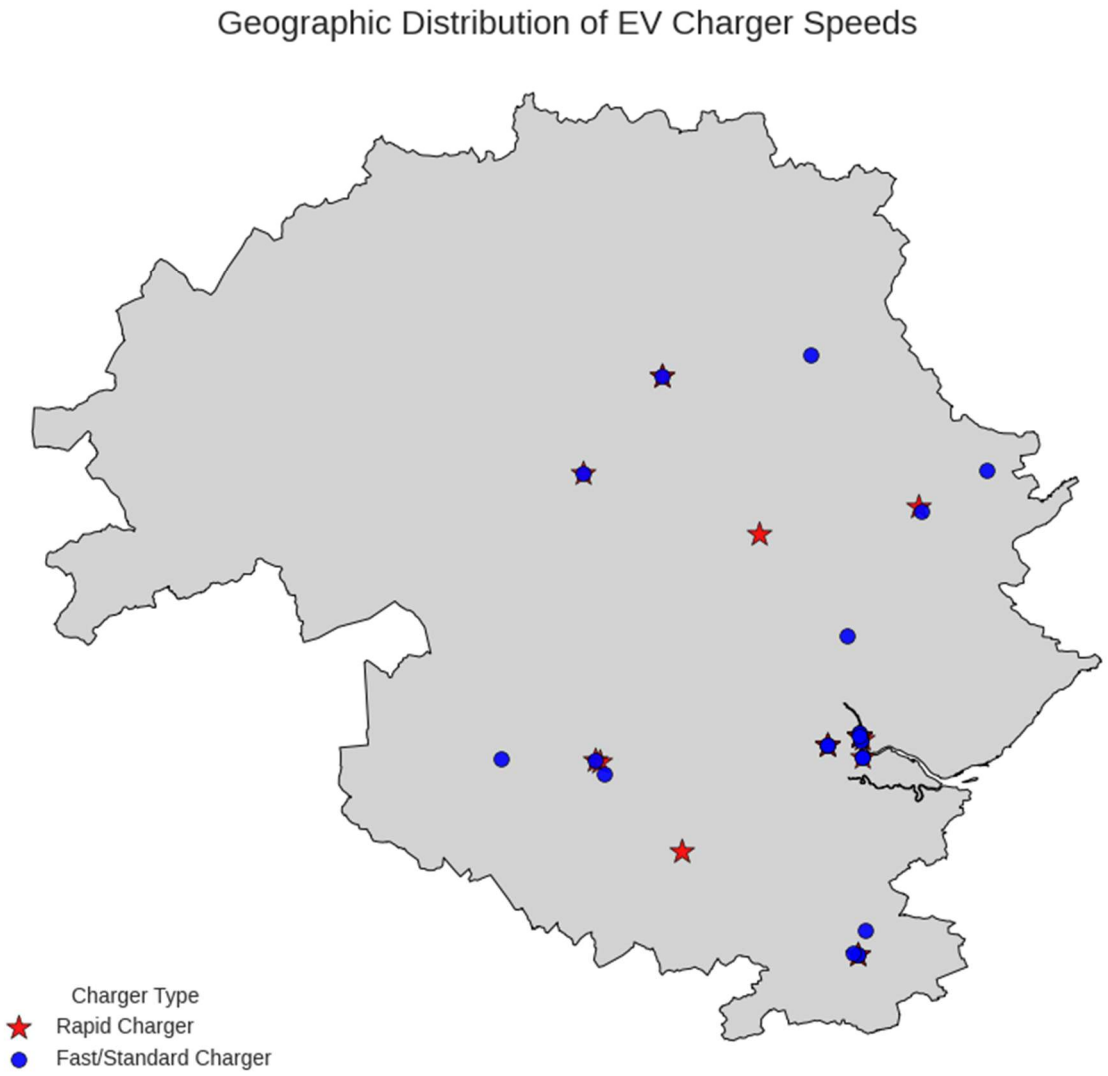


Figure 11: Geographic distribution of Rapid vs Fast/Standard EV chargers in PKC.

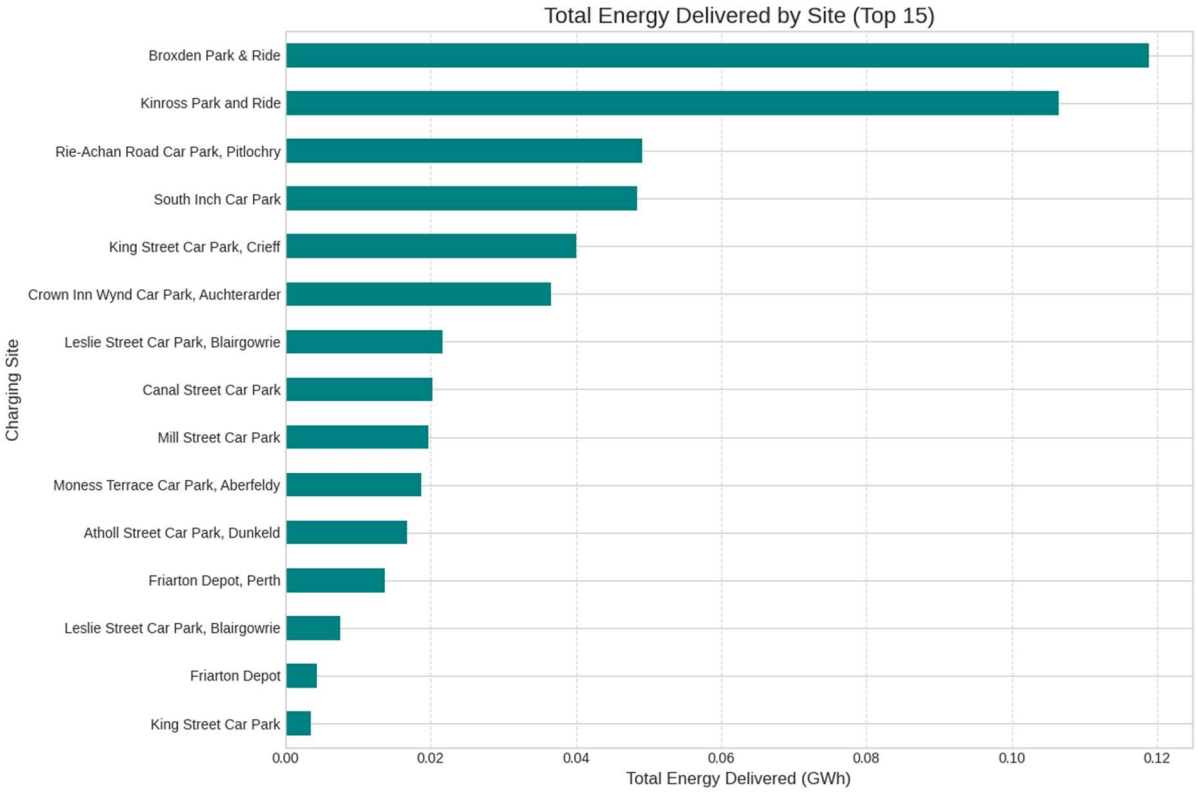


Figure 12: Total energy delivered by charging site (GWh).

4.1.2 Economic and Technological Accessibility

An examination of the network's economic and technological composition reveals key strategic choices. Figure 13 reveals that the network was predominantly "Free to Use" at the time of the data collection, confirming a policy aimed at encouraging initial EV adoption as stated in the 2022 report from PKC's Head of Planning & Development (Deans & Moran, 2022). In terms of technology, the infrastructure shows significant investment in both the modern CCS and the older CHAdeMO charging standards (Figure 14). While this demonstrates broad support for legacy vehicles, it presents a potential future-proofing challenge as the market increasingly standardises on CCS.

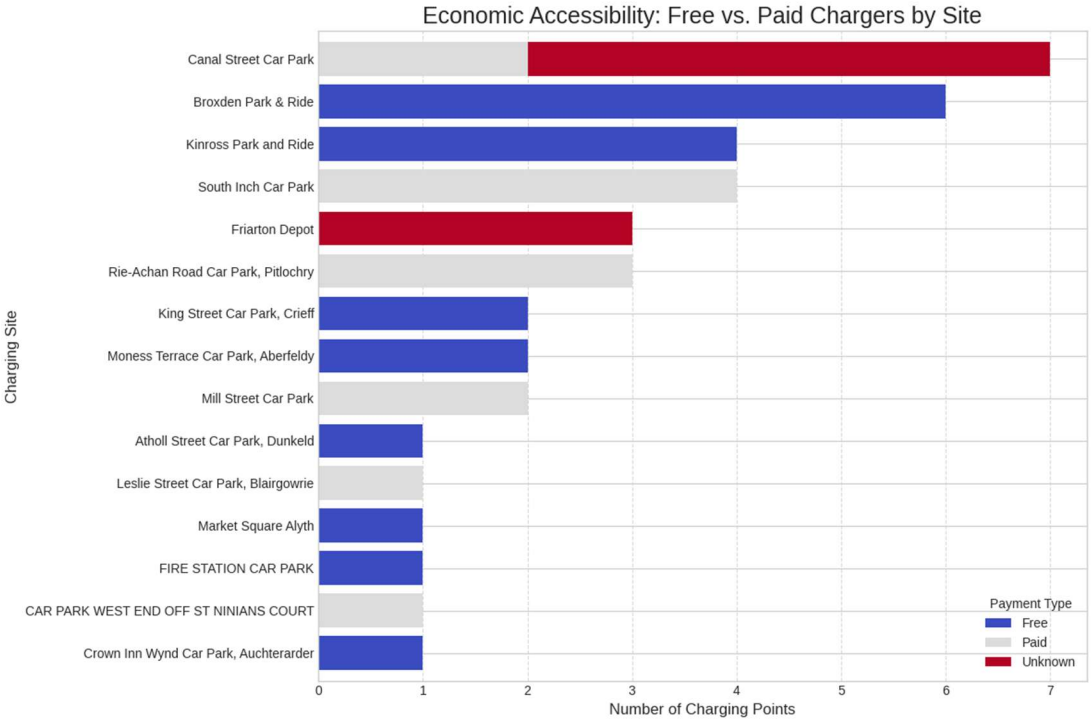


Figure 13: Economic accessibility, illustrating the distribution of free vs paid chargers by site.

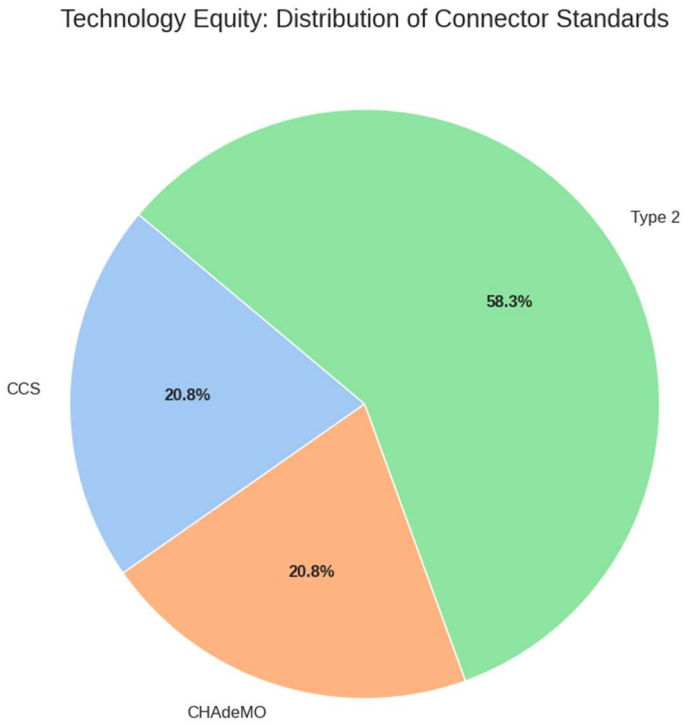


Figure 14: Distribution of charger connector standards across the network.

4.1.3 The Cost vs Convenience Trade-Off

A central challenge highlighted by this analysis is the inherent socio-technical trade-off between cost and convenience. As visualised in Figure 15, a cost-optimised schedule must shift charging away from the hours of natural user demand (represented by the "Uncontrolled Demand" profile) and into periods of lower electricity prices. This illustrates the potential for conflict between system-level economic efficiency and individual user experience, a crucial consideration for designing effective, innovative charging policies.

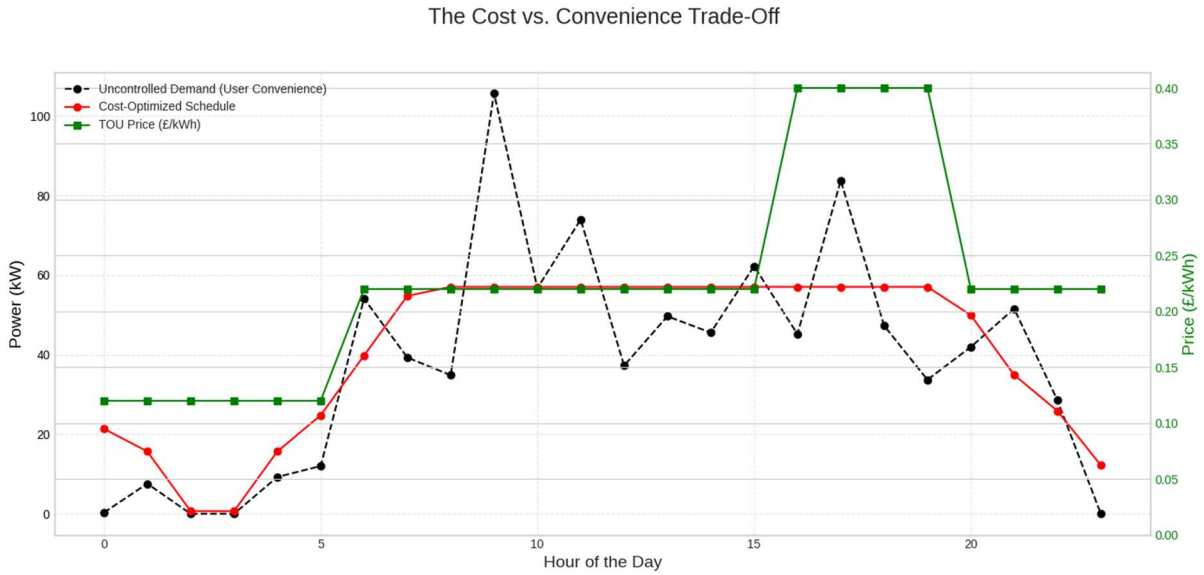


Figure 15: The Cost vs Convenience trade-off on a typical day.

4.2 Forecasting Model Performance

Following the contextual analysis of the network, four models were developed to forecast hourly EV charging demand. The performance of these models was evaluated on the unseen test set to determine the most effective approach.

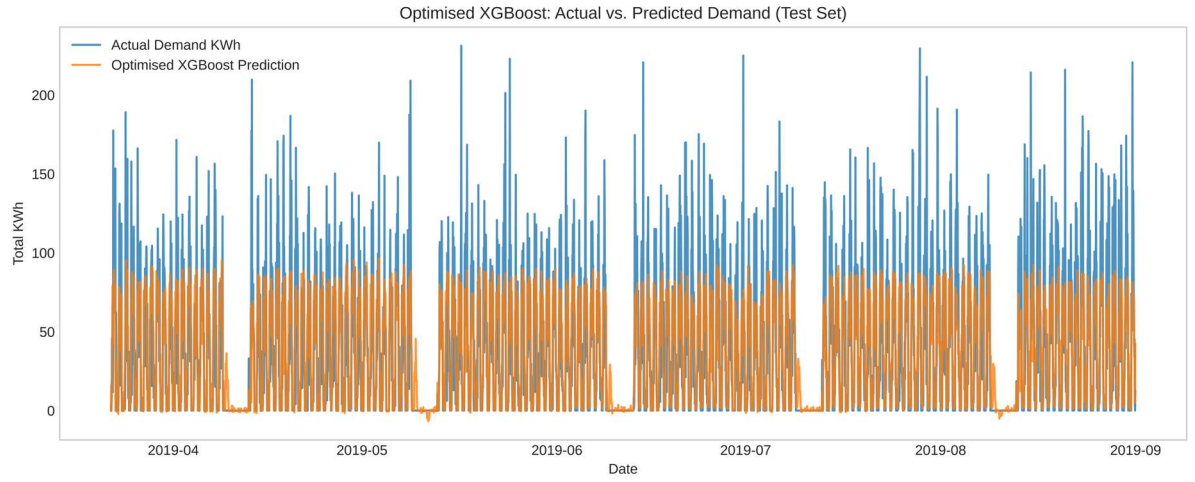
4.2.1 Comparative Evaluation of Forecasting Models

I trained four models, including multiple linear regression, SARIMA, SVR, XGBoost, and a CNN-LSTM hybrid, and evaluated them using MAE, RMSE, and R^2 on the hold-out test set (Table 1). XGBoost outperformed all others (MAE = 15.83 kWh; RMSE = 26.59 kWh; R^2 = 0.64). The full comparative analysis across all model classes, which justifies this selection, is detailed in Appendix B (Table 9). Figure 16 visually confirms its superior fit.

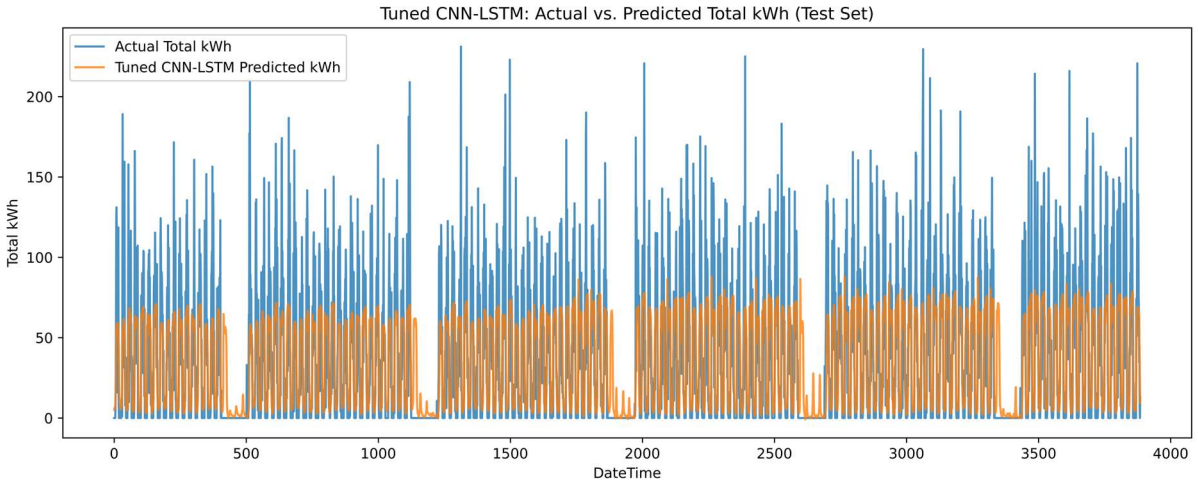
Table 1: Summary of All Model Performances on Test Set.

Model	MAE	RMSE	R^2
Optimised XGBoost	15.83	26.59	0.64
Tuned CNN-LSTM	18.43	28.97	0.56
OLS	24.05	31.29	0.49
SVR	21.55	36.44	0.31
SARIMA	24.01	36.79	0.29

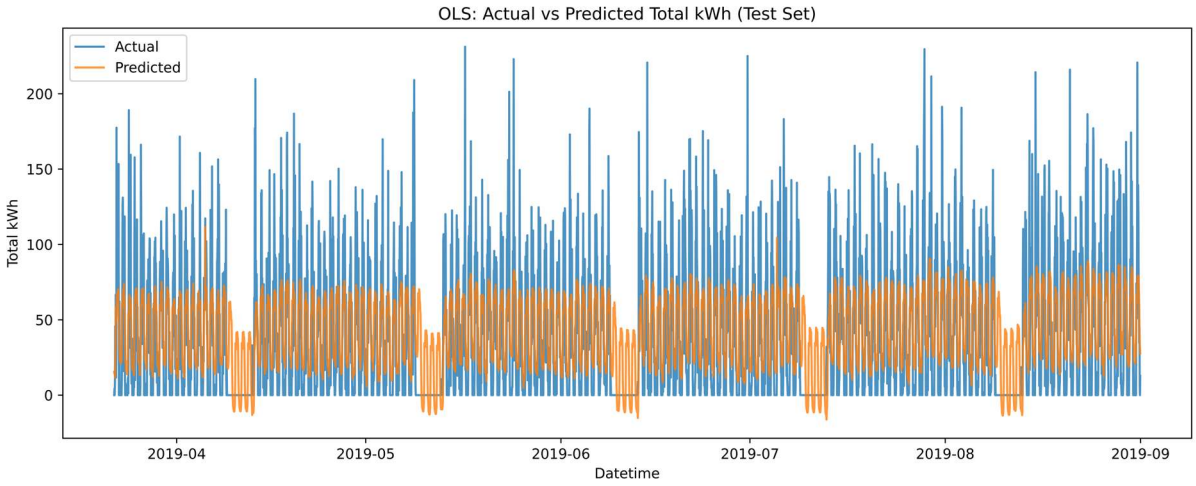
XGBoost reduces MAE by ~34% vs SARIMA, making it the most reliable model for operations.



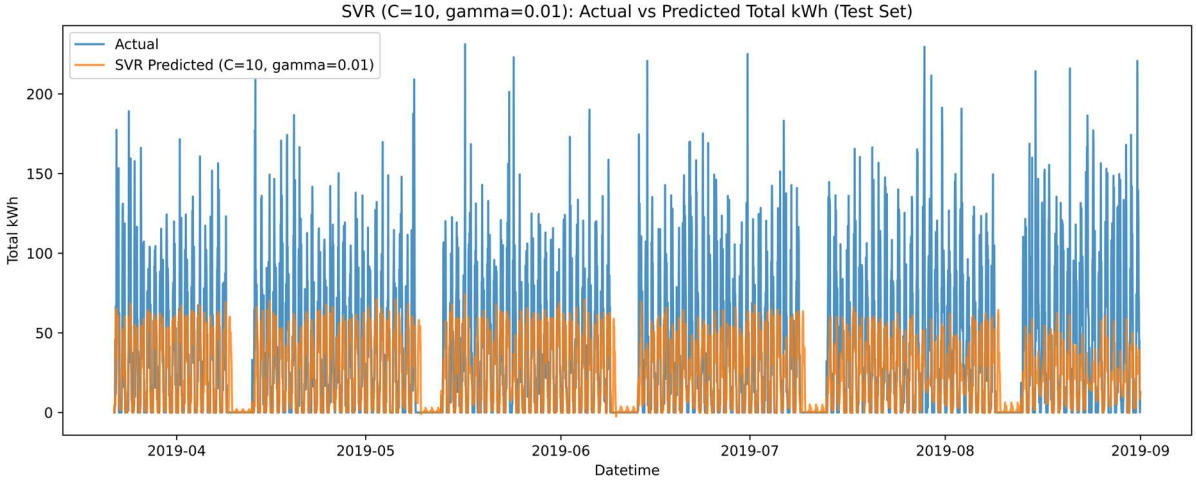
(a)



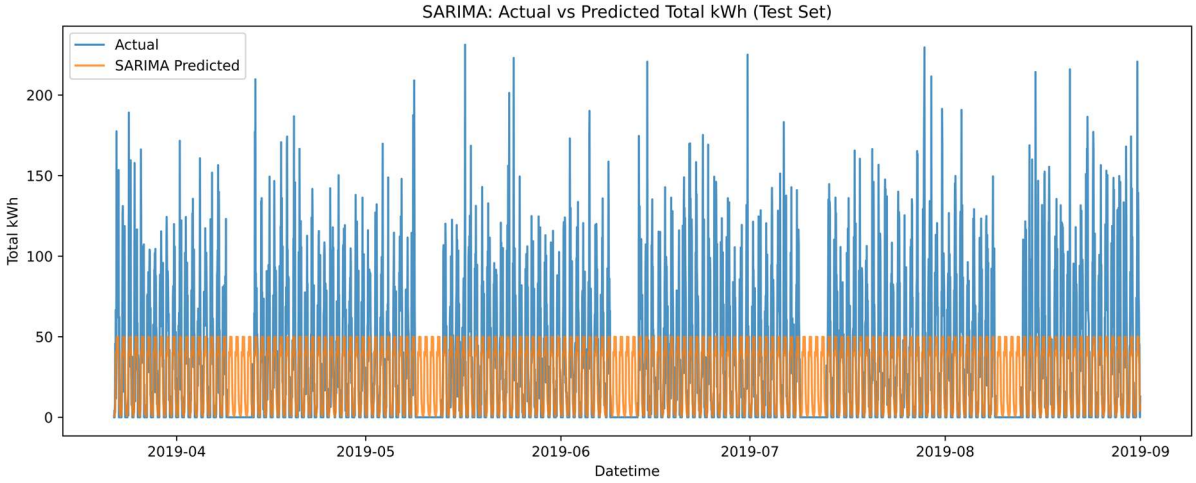
(b)



(c)



(d)



(e)

Figure 16: A comparison of actual vs predicted demand for all forecasting models.

4.2.2 Feature Importance Analysis

To interpret the estimations of the best-performing model, a feature importance analysis was conducted on the trained XGBoost model (Figure 17). The results show that `HourlyAverageDemand`, `Pavg_kW`, and `Previous24HrAverageDemand` were the most influential predictors, highlighting the strong temporal and autoregressive nature of the charging demand.

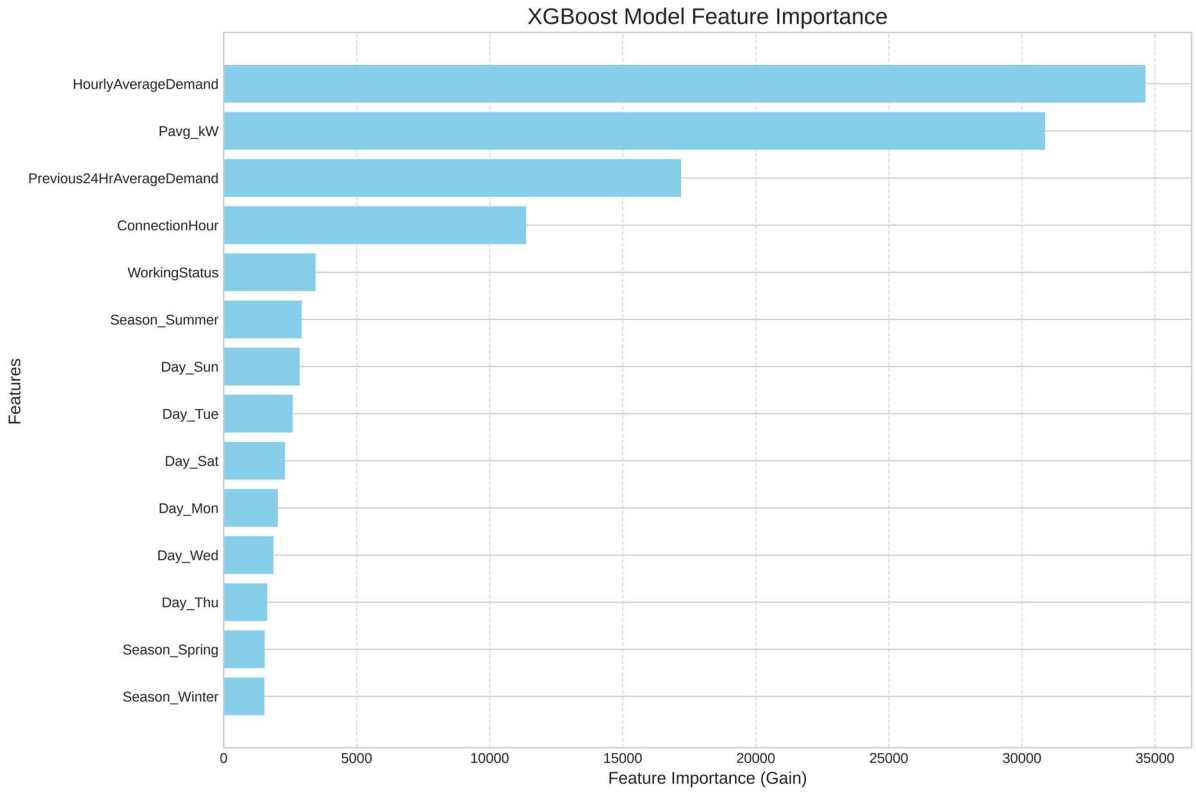


Figure 17: Bar chart of XGBoost feature importances.

4.2.3 Scenario-Specific Forecasts

Figures 18–21 compare XGBoost and CNN-LSTM forecasts with actual demand for a weekday, weekend day, a whole week, and a holiday week. The result shows that XGBoost consistently captures peak shifts and troughs more accurately, especially under atypical demand conditions. More scenario-based forecasts are shown in Appendix G.

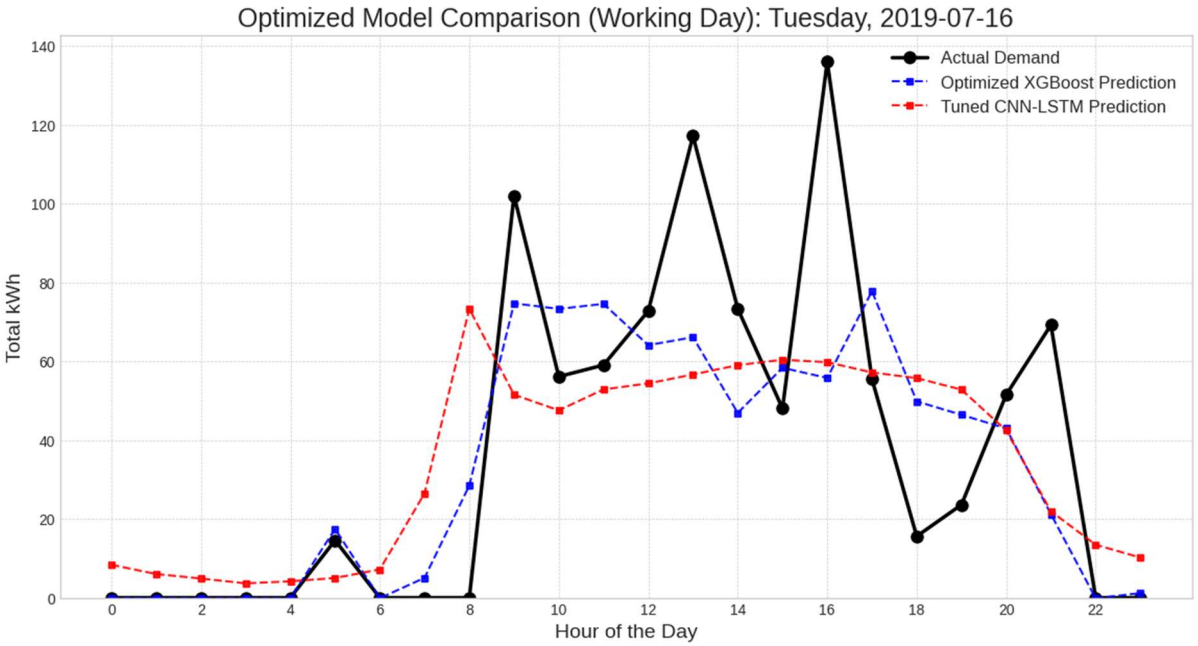


Figure 18: A comparison of XGBoost and CNN-LSTM forecasts on a typical working day.

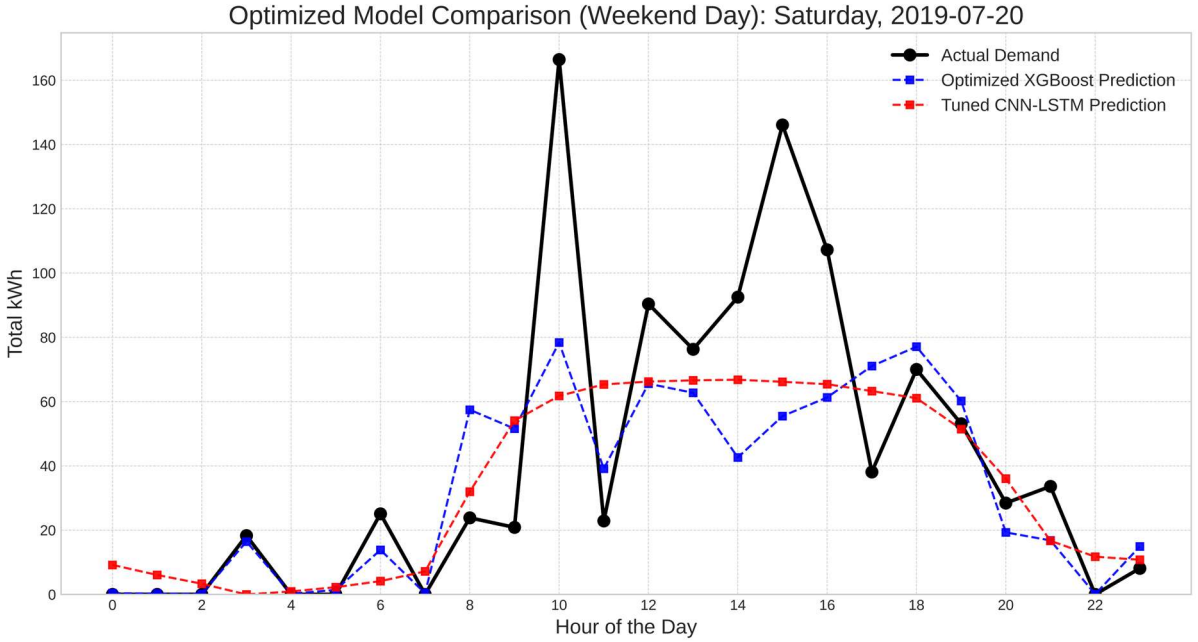


Figure 19: A comparison of XGBoost and CNN-LSTM forecasts on a typical weekend.

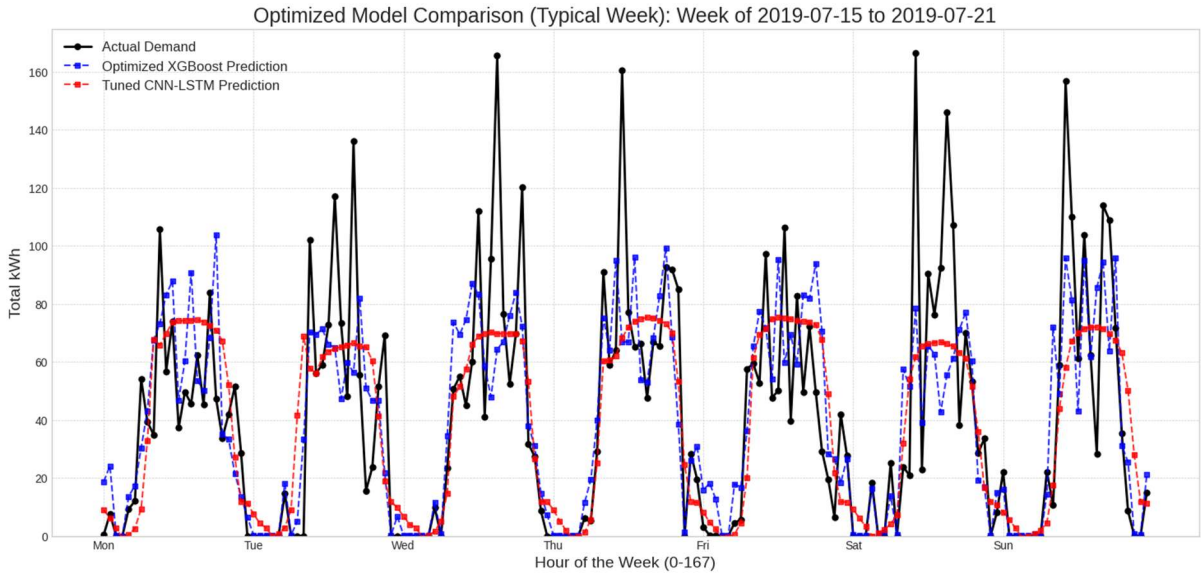


Figure 20: A comparison of XGBoost and CNN-LSTM forecasts on a typical week.

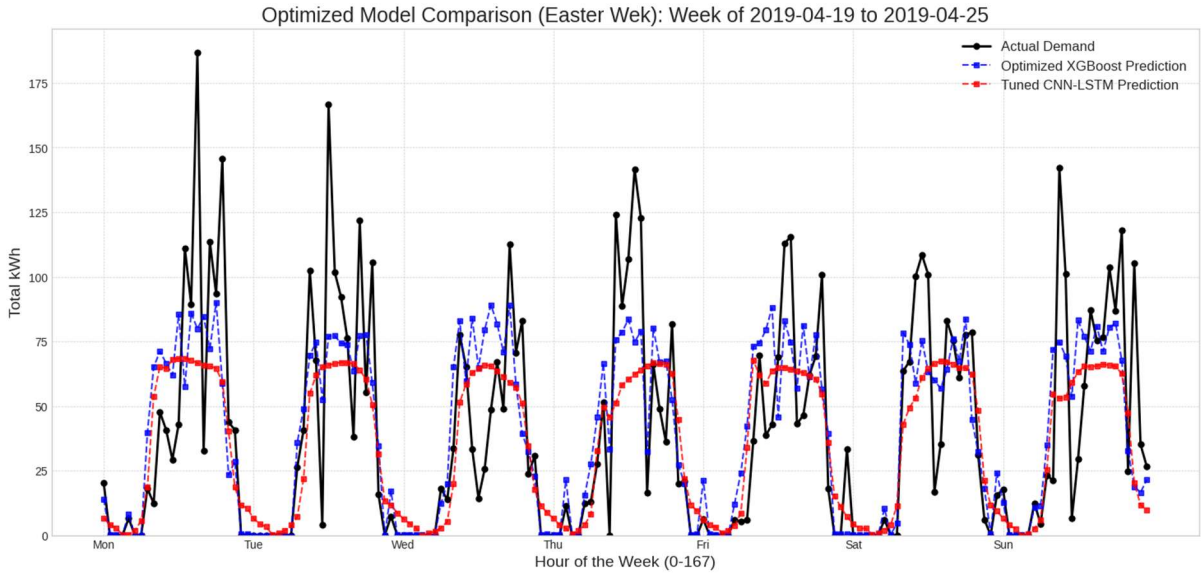


Figure 21: A comparison of XGBoost and CNN-LSTM forecasts on a holiday week.

4.2.4 Uncertainty-Aware Forecasting

To support an optimisation strategy that accounts for uncertainty, the ST-UOF framework extends the XGBoost model via quantile regression to generate 10^{th}, 50^{th} and 90^{th} percentiles. Figures 22 and 23 illustrate that the resulting prediction intervals reliably encompass observed demand volatility, forming the uncertainty bounds for the optimisation.

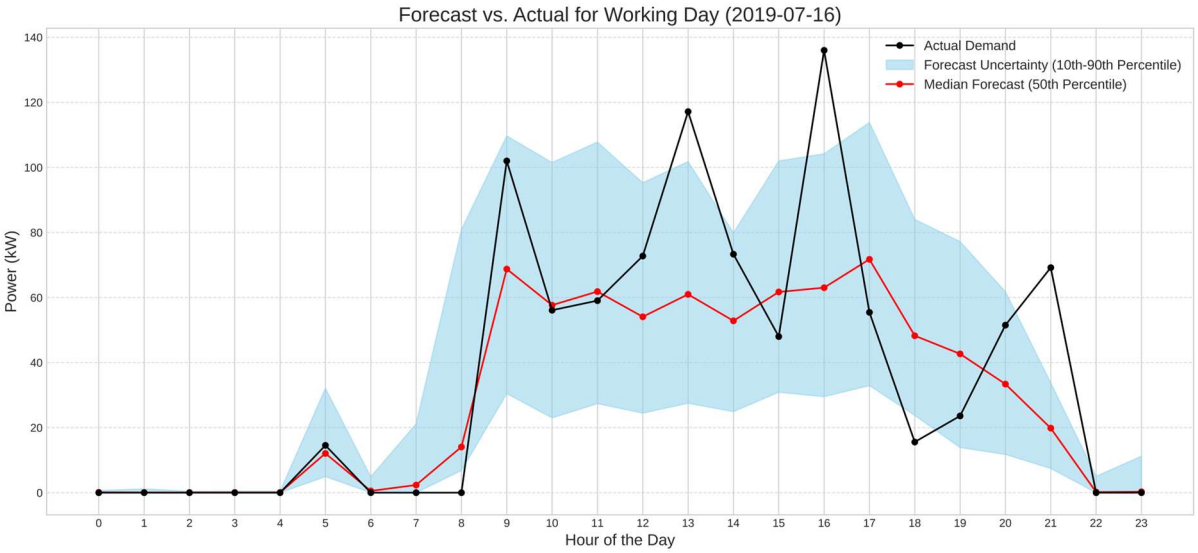


Figure 22: An uncertainty-aware forecast for a typical day with prediction bands.

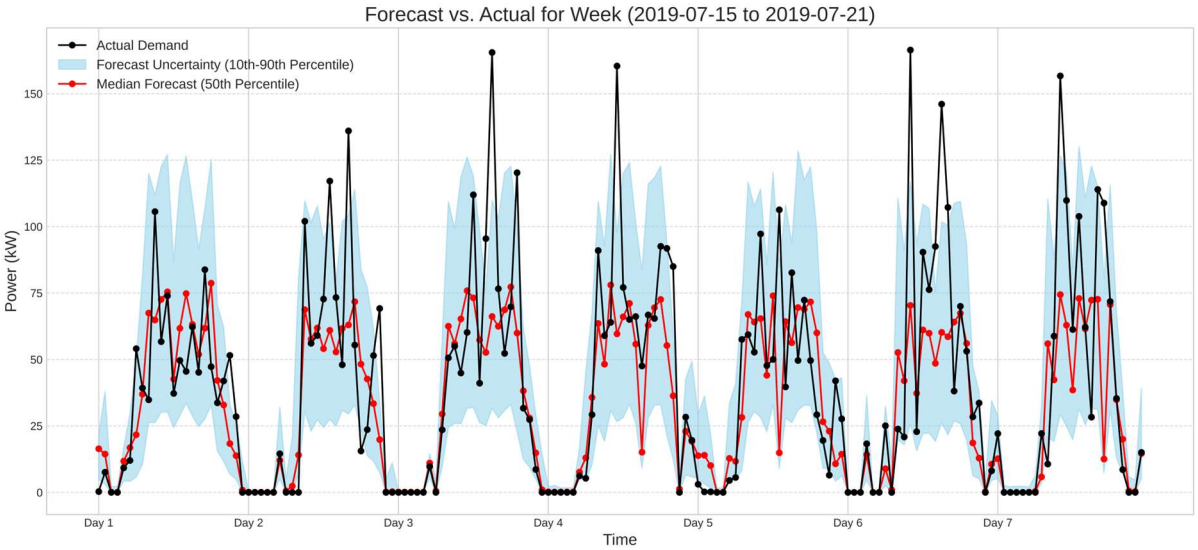


Figure 23: An uncertainty-aware forecast for a typical week with prediction bands.

4.3 Performance of Charging Optimisation Strategies

The final analysis phase simulated three charging strategies – Uncontrolled, Rule-Based, and Forecast-Informed Optimisation – to quantify their operational performance and potential cost savings.

4.3.1 Optimisation Results

The forecast-informed optimiser produces a 24-hour schedule that minimises energy plus demand-charge cost while respecting physical limits and an hourly uncertainty cap. Figure 24 shows the median forecast (dashed blue), the 10–90% band (shaded), the TOU tariff, and two LP schedules: one without a ramp limit and one with a ±15 kWh per hour ramp. The

unconstrained plan jumps early to a flat, cheap level; the ramp-limited plan climbs and descends smoothly, still steering away from the 16:00–19:00 peak.

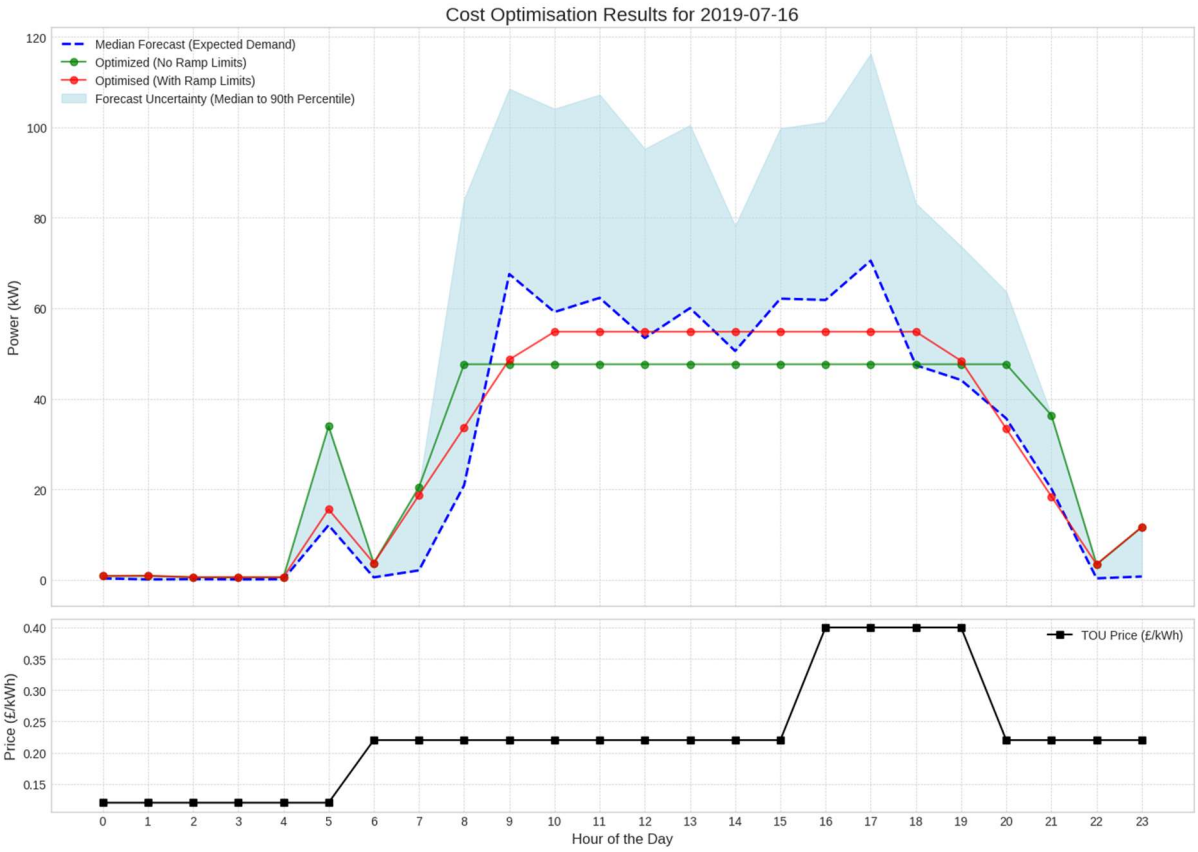
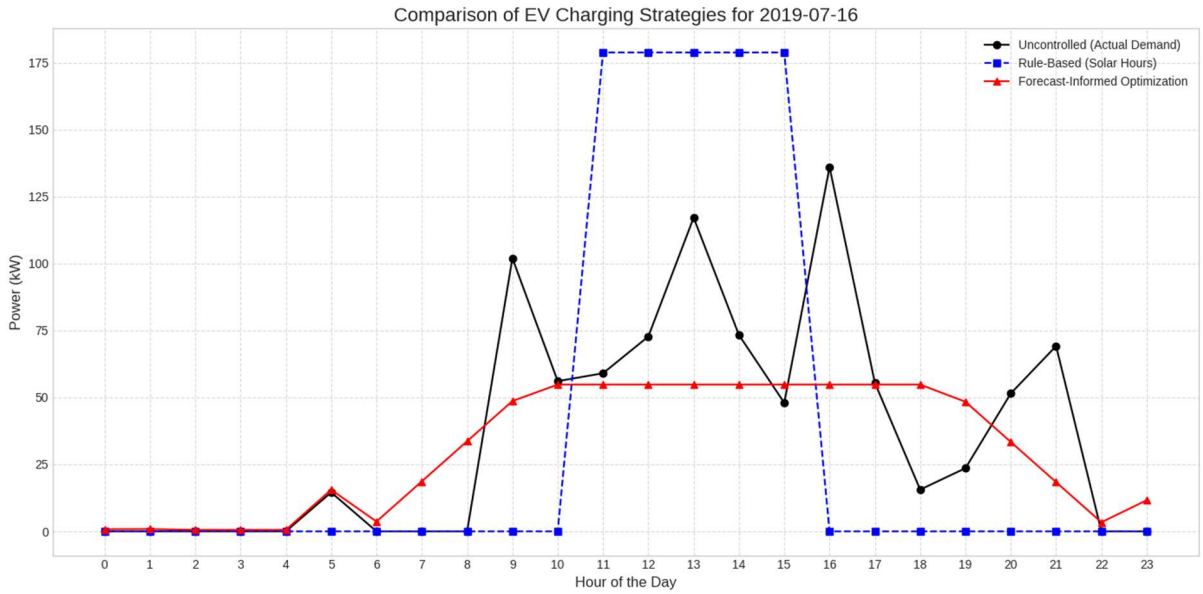


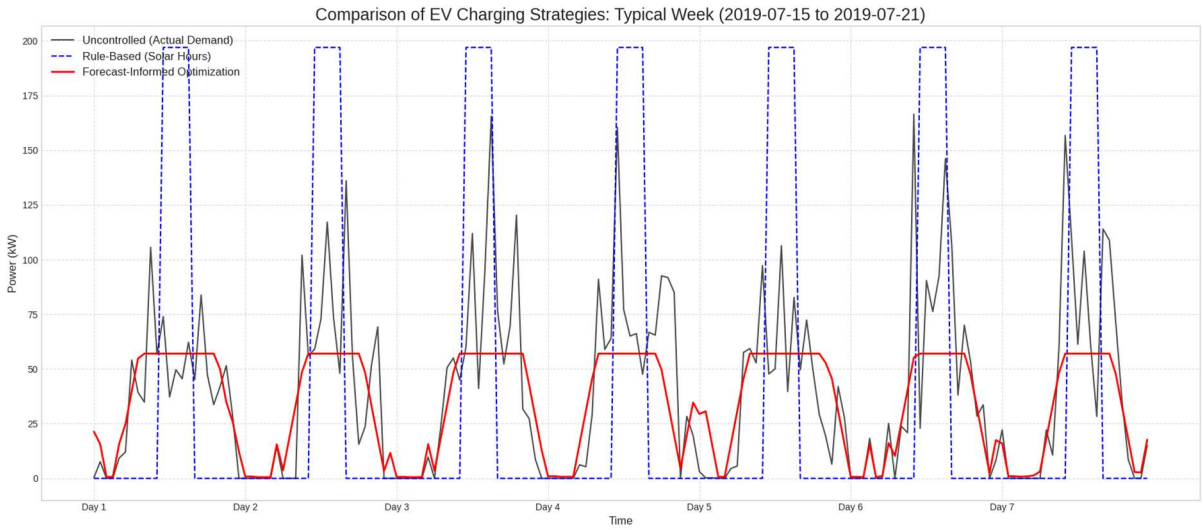
Figure 24: Median forecast, 10–90% band (cap), TOU tariff, and LP schedules with/without the ramp limit.

4.3.2 Comparative Strategy Analysis

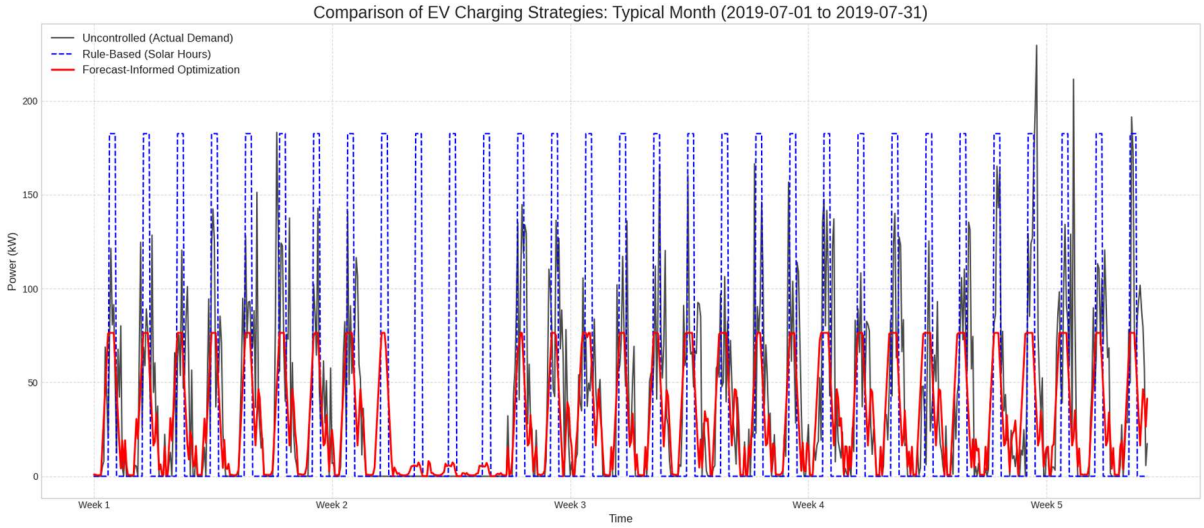
To quantify these effects across days, weeks, and months, the forecast-informed schedule smooths load and shifts energy into low-price hours; the rule-based window creates spikes, and the uncontrolled baseline remains volatile (Figure 25). Tables 2–4 report the cost split, showing that savings reach 53.2% for the day case and remain substantial at 41.7% (week) and 38.0% (month), primarily driven by lower demand charges. Tariffs follow a three-block TOU (overnight, day, peak with a 16:00–19:00 peak). Varying the demand charge, TOU spread, and cap quantile leaves the direction and size of savings broadly unchanged, as demand-charge mitigation dominates; see the sensitivity analysis in Appendix C (Tables 11–13). Additionally, a simple “off-peak window” baseline concentrates energy into a few hours, triggering high demand charges; Appendix C (Table 14) shows negative savings in cases with low or baseline spreads.


Comparison of EV Charging Strategies for 2019-07-16
Uncontrolled (Actual Demand)
Rule-Based (Solar Hours)
Forecast-Informed Optimization
Power (kW)
Hour of the Day

(a)


Comparison of EV Charging Strategies: Typical Week (2019-07-15 to 2019-07-21)
Uncontrolled (Actual Demand)
Rule-Based (Solar Hours)
Forecast-Informed Optimization
Power (kW)
Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
Time

(b)



(c)

Figure 25: A comparison of the three charging strategy profiles over daily, weekly, and monthly horizons.

Table 2: Cost and Savings Comparison for a Sample Day.

Strategy	**Energy Cost (£)**	**Demand Charge (£)**	**Total Cost (£)**	**Cost Savings (£)**	**Savings (%)**
Uncontrolled	236.78	1,631.98	1,868.75	0.00	0.0
Rule-Based	196.72	2,146.06	2,342.79	-474.03	-25.4
Forecast-Informed	202.10	672.54	874.64	994.12	53.2

Table 3: Cost and Savings Comparison for a Sample Week.

Strategy	**Energy Cost (£)**	**Demand Charge (£)**	**Total Cost (£)**	**Cost Savings (£)**	**Savings (%)**
Uncontrolled	1,843.30	1,997.23	3,840.53	0.00	0.0
Rule-Based	1,516.54	2363.43	3,879.97	-39.44	-1.0
Forecast-Informed	1,545.61	692.56	2,238.17	1,602.36	41.7

Table 4: Cost and Savings Comparison for a Sample Month

Strategy	Energy Cost (£)	Demand Charge (£)	Total Cost (£)	Cost Savings (£)	Savings (%)
Uncontrolled	7,657.70	2,754.38	10,412.08	0.00	0.0
Rule-Based	6,222.94	2,189.89	8,412.84	1,999.24	19.2
Forecast-Informed	5,554.43	904.56	6,458.99	3,953.09	38.0

4.4 Chapter Summary

This chapter presented the empirical findings. The socio-technical analysis revealed a network clustered around a few central sites, with fewer and smaller sites on the edges. It highlighted potential gaps in coverage and long-term planning. The forecasting evaluation identified XGBoost as the most accurate model. Using those forecasts, the optimiser produced schedules that met site caps and ramp limits, delivering consistent cost savings over uncontrolled and simple rule-based strategies across day, week, and month. The next chapter interprets these results, considering the research questions and prior work.

Chapter 5: Discussion

This chapter provides a critical interpretation of the empirical findings presented in Chapter 4, synthesising them with the theoretical framework and research gaps established in the literature review. The discussion is structured to move from the specific technical results to their broader socio-technical implications, creating a cohesive argument that directly addresses the dissertation's central research questions and hypotheses. It begins by critically evaluating the forecasting model performance in response to the secondary research question (RQ2), followed by an analysis of the optimisation results as they pertain to the primary research question (RQ1). Finally, it synthesises these technical findings with the socio-technical context established in the exploratory analysis, using socio-technical transition theory as a lens to examine the real-world implications for equitable and sustainable EV infrastructure planning.

5.1 Forecasting Performance and the Importance of Socio-Temporal Features (RQ2)

RQ2 asked whether machine learning models significantly outperform traditional statistical models for this forecasting task. The results provide a clear and affirmative answer, strongly supporting Hypothesis 2 (H2) and offering a nuanced critique of the role of model complexity in this specific socio-technical context.

5.1.1 Forecasting Performance

The quantitative evaluation presented in Table 1, and further detailed in Appendix B, confirms that the XGBoost model delivered a significant improvement in accuracy, reducing the Mean Absolute Error (MAE) from 24.01 for the SARIMA model to 15.83 – a reduction of approximately 34%. This outcome empirically validates the argument made in the literature review that the limitations of traditional statistical models – namely, their reliance on linear assumptions – make them ill-suited for capturing the complex dynamics of charging behaviour (Zhou et al., 2025).

More critically, the XGBoost model also outperformed the more complex hybrid CNN-LSTM (MAE of 18.43), suggesting that for this type of problem, the value of feature engineering that captures human routines outweighs the architectural complexity of deep learning. This aligns with findings that highlight the strength of gradient boosting on tabular datasets where feature diversity is high (Chen & Guestrin, 2016; Friedman, 2001).

However, while XGBoost proved most accurate, qualitative analysis reveals its limitations. The model still exhibited residual errors, particularly during volatile peak hours, reflecting the inherently stochastic nature of individual decisions. This difficulty in capturing the sharpest peaks may be attributed to data drift, as the model was primarily trained on data from 2016 to 2018, while the test set was from 2019 – a period with higher demand and likely different charging behaviours (Figure 7). Despite this, the model's ability to adapt to atypical demand patterns, such as the public holiday scenarios shown in Appendix E, demonstrates its real-world robustness. Also, more advanced architectures – such as Transformer models for temporal attention or Graph Neural Networks (GNNs) to capture spatial interdependencies between charging sites – could have been explored to address the extremes. Nevertheless, these architectures demand substantially larger datasets and greater computational resources; their deployment should be weighed against the incremental accuracy gains and the scale at which local authorities operate (Kipf & Welling, 2017; Vaswani et al., 2023).

This finding directly addresses the critique of deterministic forecasting in the literature review (Fan et al., 2023). The success of the quantile regression intervals in capturing this volatility (Figures 22 and 23) demonstrates that the research gap concerning the under-utilisation of probabilistic methods is not merely theoretical, but a necessary step. The statistical reliability of these intervals was further validated through a forecast calibration analysis, detailed in Appendix F. For operational decision-making, a realistic estimation of uncertainty is arguably more valuable than marginal gains in point forecast accuracy. In summary, the findings support H1, but also critically position probabilistic forecasting as a necessary evolution beyond simple accuracy metrics. However, this call for probabilistic methods must be tempered by practical challenges of implementation, such as computational complexity and usability issues for local authorities with limited resources (Logan et al., 2021; Rashid et al., 2024).

5.1.2 The Social Drivers of Demand: An Analysis of Feature Importance

The feature importance analysis (Figure 17) provides a data-driven confirmation of socio-technical transition theory, which posits that outcomes are shaped by the interplay of technology, policy, and user behaviour (Geels, 2019). The model's performance was overwhelmingly driven by socio-temporal indicators (`WorkingStatus, ConnectionHour`) and autoregressive features (`Previous24HrAverageDemand`). This is a direct empirical demonstration that the "social" dimension – the predictable, rhythmic patterns of human life – is a dominant driver of the "technical" system's behaviour (Mckenna & Thomson, 2016). Also,

these findings are corroborated by recent industry analysis. For instance, the National Grid ESO's 'Future Energy Scenarios 2025' report identifies driver behaviour as a primary source of forecast uncertainty (National Grid ESO, 2025), this reinforces the critical need for the probabilistic forecasting methods developed in this research.

5.2 The Value of Optimisation and the Cost-Convenience Dilemma (RQ1)

RQ1 examined the operational and cost-saving benefits of a forecast-informed optimisation strategy. The simulation results provide strong evidence in support of Hypothesis 1 (H1), demonstrating substantial savings while simultaneously quantifying a fundamental socio-technical conflict.

5.2.1 The Economic Case for Smart Charging: Deconstructing the Savings

The analysis provides a compelling economic case for smart charging. Crucially, it reframes the primary value proposition. While much of the literature discusses energy arbitrage, the results in Table 2 show that the savings are almost entirely driven by the reduction of demand charges, which heavily penalise peak usage under UK commercial tariffs (Energy Networks Association, 2025).

By smoothing the aggregate load profile seen in Figure 25, the forecast-informed schedule achieved a daily cost saving of £994.12 (a 53.2% reduction). A comprehensive sensitivity analysis, detailed in Appendix C, confirmed that these savings are robust even when key financial parameters, such as the demand charge and TOU price differential, are varied. Demand-charge mitigation accounted for approximately 96% of the benefit. This finding provides a critical insight for local authorities: the most significant financial return from smart charging comes from managing peak demand. This finding addresses the operational pressures identified in Chapter 1.

5.2.2 The Cost vs Convenience Trade-Off: A Socio-Technical Dilemma

While the economic case is clear, a critical analysis of the results reveals a core socio-technical dilemma. The "Cost vs Convenience Trade-Off" plot (Figure 15) visually confirms the tension between system-level optimisation and individual user needs, as the cost-optimal schedule is often in direct opposition to the convenience-driven, uncontrolled demand profile. To achieve savings, the system must defer charging away from morning and evening peaks – precisely when users are most likely to want to charge. This is an illustration of socio-technical theory

in action, where a purely technical, cost-optimal solution directly conflicts with established user behaviour.

This study, therefore, not only confirms the existence of this trade-off but quantifies it, providing empirical weight to the argument that a purely technical solution is socially unsustainable without effective "socio" interventions like behavioural nudges and dynamic pricing (Octopus Energy, n.d.). This challenge is recognised at a national level; the UK Government's and Ofgem's joint EV Smart Charging Action Plan acknowledges that making smart charging the "convenient choice" is a primary barrier to overcome (GOV.UK, 2023).

Ultimately, while the results confirm that forecast-informed optimisation can substantially reduce operational costs, this technical success is not socially neutral. A critical interpretation suggests that its real-world success hinges on resolving this cost-convenience dilemma. The success of any innovative charging system will therefore depend not just on its algorithms, but on its ability to incorporate behavioural incentives and user-friendly interfaces that bridge the gap between economic efficiency and the practical needs of drivers.

5.3 Synthesising Findings: Optimisation as a Socio-Technical Actor in the UK Context

Having answered the primary research questions, this final section synthesises the technical findings with the socio-technical context, critically examining how the proposed technological solution itself can become an actor within the system, shaping future outcomes and potentially reinforcing the very inequalities it should aim to solve, particularly considering current UK policy. A core tenet of socio-technical theory is that technology is not neutral; it actively shapes social structures (Geels, 2019). The optimisation algorithm developed here is a prime example. Its objective function – to minimise cost – makes it an active force that, if implemented naively, would reshape the charging landscape based on a purely economic logic.

5.3.1 Geographic and Economic Equity

The exploratory analysis revealed a centralised network pattern – where infrastructure is concentrated in central hubs – which mirrors the persistent national issue of "charging deserts." This is not a historical problem; a recent Public Accounts Committee report reiterated this geographical divide, noting that 43% of public chargers are concentrated in London and the Southeast. In response, the UK Government's primary policy is the Local Electric Vehicle Infrastructure (LEVI) Fund, which aims to address this imbalance.

However, a purely economic optimisation, such as the cost-minimising algorithm developed in this research, creates a direct policy conflict. The algorithm would naturally generate the greatest savings by managing demand at the most heavily used sites (Figure 12), thereby concentrating efficiency gains in already well-served hubs. At the same time, the LEVI fund pushes for expansion into less commercially viable areas. Such an approach risks creating a feedback loop where high-use sites receive the most sophisticated management, enhancing their appeal while LEVI-funded rural sites are neglected, cementing their marginal status. This problem is compounded by the shift to paid charging (Figure 13), as the algorithm would further prioritise revenue-generating locations, potentially creating a two-tiered system where optimised charging is primarily available in already accessible hubs.

5.3.2 Technological Equity and Future-Proofing

This tendency for a cost-minimising algorithm to prioritise the most heavily used assets extends to technological equity. The analysis of connector standards (Figure 14) showed a mix of modern CCS and legacy CHAdeMO assets. An optimisation algorithm will inevitably prioritise the most efficient and popular chargers – likely the CCS hubs – to maximise its impact on the network's peak load. This could further marginalise owners of older vehicles reliant on CHAdeMO, effectively using a technical solution to accelerate the obsolescence of certain user groups' assets.

In each of these dimensions, the optimisation algorithm ceases to be a simple tool and becomes a powerful socio-technical actor. Without explicit policy intervention to embed equity goals into its objective function – for instance, by adding constraints to ensure a minimum service level in LEVI-funded zones or protect legacy assets – a purely technical optimisation risks reinforcing, rather than alleviating, the very inequalities that current UK policy is trying to solve.

5.3.3 Intended Use and the Cost-Convenience Trade-Off

Finally, the model's network-level perspective risks overlooking the specific Intended Use of individual sites. The exploratory analysis identified chargers with different social functions, from short-stay "shopper" chargers to long-stay "commuter" hubs. The Cost-Convenience Trade-Off discussed previously, therefore, does not manifest uniformly across these sites, creating a potential mismatch between the technical solution and the social function of the asset.

This conflict is best illustrated with examples. For a commuter at a long-stay hub, a cost-driven schedule that defers charging to cheap, off-peak overnight hours might be perfectly acceptable. However, for a driver needing a quick top-up at a short stay "shopper" charger, any delay imposed by the optimiser would represent a significant inconvenience, directly conflicting with the site's intended purpose. A purely cost-driven optimisation might even schedule heavy charging at a commuter car park during midday to absorb cheap solar energy, even if the intended users are at work elsewhere.

This mismatch between optimised schedules and user intent will become more acute as new regulations, which came into force in late 2024, mandate 99% reliability for the rapid charging network (The Public Charge Point Regulations, 2024). This policy increases pressure on operators to maximise uptime and availability, especially at high-traffic sites where convenience is paramount.

5.4 Chapter Summary

This chapter critically interpreted the research findings, directly addressing the dissertation's research questions. It confirmed that machine learning models significantly outperform traditional forecasting methods (RQ2) but highlighted that probabilistic approaches are paramount for capturing real-world uncertainty. The findings also validated that forecast-informed optimisation offers substantial economic benefits (RQ1), while quantifying the inherent socio-technical conflict between system cost and user convenience. Integrating these results through a socio-technical lens, the discussion argued that a successful EV transition requires technical solutions designed to address, not amplify, social and spatial inequalities.

Chapter 6: Conclusion

This dissertation set out to address the intertwined technical and social challenges of managing public Electric Vehicle (EV) charging infrastructure. Grounded in a real-world case study of Perth and Kinross Council (PKC), the research developed and evaluated a forecast-informed optimisation framework, demonstrating that a holistic, socio-technical approach is essential for the sustainable and equitable deployment of EV charging networks.

6.1 Summary of Findings and Answering Research Questions

The empirical investigation confirmed both primary hypotheses. Answering RQ1, the study confirmed that a forecast-informed optimisation strategy delivers substantial economic benefits, quantifying daily cost savings of over 53%, primarily driven by mitigating peak demand charges, which provides a strong economic case for intelligent management systems. Answering RQ2, the research established that an XGBoost model using socio-temporal features significantly outperformed traditional statistical models, and that its accuracy was enhanced with uncertainty quantification via quantile regression. This provides the robust, probabilistic forecasts necessary for operational planning.

6.2 Main Contribution to Literature

The primary contribution of this dissertation is the development and validation of the Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF), an integrated, end-to-end methodology. Unlike studies that treat forecasting, optimisation, and social analysis as separate domains, this research has demonstrated a cohesive pipeline where the initial socio-technical analysis of the network's equity and design directly motivates and informs the subsequent technical modelling. By combining a robust, uncertainty-aware forecasting engine – whose statistical reliability was validated in Appendix D through coverage analysis and pinball loss metrics – with a constrained optimisation model, this work offers a practical framework for creating more resilient and financially sustainable innovative charging systems.

6.3 Limitations of the Study

The most significant limitation is the age of the data and the obscurity of the methodology used for collection. The model is based on behaviour from 2016–2019 sessions, which were largely early adopters, hence introducing demographic bias. These users were wealthier on average, more likely to have home charging, and perhaps more forgiving of a patchy network. It follows that the model was trained on a narrow slice of users, and the clean daily and weekly rhythms

it highlights may matter less in today's broader, more price-sensitive market. Also, due to the lack of a record on the data collection methodology, the reliability and completeness of the data cannot be independently verified at the time of this study.

The analysis also took pragmatic shortcuts. Historical records are treated as a proxy for unmanaged demand; if any quiet load management was already in place, the savings estimates are likely conservative. Tariffs were simplified for clarity, which helps exposition but flattens the messy reality of volatile prices. Furthermore, results are reported at a network scale, which is useful for strategy but cannot capture queues at a given site or the needs of a driver far from alternatives.

Two further points temper the reported effect sizes. First, the savings are calculated against a simple fixed-window rule rather than a more sophisticated time-of-use heuristic. Against a stronger TOU rule, the savings margin would likely shrink; therefore, the results should be interpreted as an upper bound relative to naïve operation. Second, although the main analysis employs a fixed set of parameters for clarity, a comprehensive sensitivity analysis was conducted to assess the robustness of these findings (Appendix C). This analysis confirmed that the core conclusions remain intact when key settings – including demand-charge levels, the TOU price spread, and the forecast-uncertainty cap – are varied.

Finally, the equity discussion is descriptive rather than metric-driven. An accessibility index, such as population-weighted drive-time to the nearest high-power charger, was not computed. The "charging deserts" reading is therefore qualitative and should be treated with caution.

6.4 Recommendations for Future Work

Future work should broaden the data and context and deepen operational modelling. First, replace early-adopter bias by pairing updated usage data with short surveys of current users to capture their motives and barriers. Then, test ST-UOF in a dense urban area, such as a London borough, to gauge its generalisability. Second, move beyond a network aggregate: a graph neural network can model spatial coupling and spillover between sites to deliver station-level forecasts. Third, enhance the optimiser: enrich forecasts with exogenous signals, such as weather and traffic, and extend the scheduling model to include battery energy storage, allowing the system to shift and shave peaks more effectively. Together, these steps would test transferability and lift both accuracy and actionable savings.

6.5 Recommendations for Industry

The findings suggest recommendations for both public policy and private industry. For local authorities, this means continuing targeted investment in underserved "charging deserts" while future-proofing high-traffic hubs with modern connectors. For industry actors, such as charge point operators and software developers, the ST-UOF provides a framework for developing next-generation innovative charging solutions. Crucially, both groups must prioritise user convenience in the design of these schemes to ensure they are embraced by the public.

6.6 Personal Reflection

On a personal level, this research has been a profound learning experience, solidifying the critical importance of a "socio-technical first" approach described by Trist & Bamforth (1951). It has shown me that the goal of data science is not just to build accurate models, but to generate actionable insights that can inform better, more equitable, and more sustainable real-world decisions.

Ultimately, this dissertation argues that EV infrastructure challenges are not purely technical but are deeply embedded in the social, economic, and behavioural patterns of the communities they serve. The findings show that while machine learning is a powerful tool, its value is only realised within a framework that acknowledges real-world constraints and equity. A successful and just transition to decarbonised transport requires not just more innovative technology, but a smarter, more integrated, and human-centred approach to its implementation.

References

Agga, A., Abbou, A., Labbadi, M., Houm, Y. E., & Ou Ali, I. H. (2022). CNN-LSTM: An efficient hybrid deep learning architecture for predicting short-term photovoltaic power production. *Electric Power Systems Research*, *208*, 107908. https://doi.org/10.1016/j.epsr.2022.107908

Akshay, K. C., Grace, G. H., Gunasekaran, K., & Samikannu, R. (2024). Power consumption prediction for electric vehicle charging stations and forecasting income. *Scientific Reports*, *14*(1), 6497. https://doi.org/10.1038/s41598-024-56507-2

Akter, S., McCarthy, G., Sajib, S., Michael, K., Dwivedi, Y. K., D'Ambra, J., & Shen, K. N. (2021). Algorithmic bias in data-driven innovation in the age of AI. *International Journal of Information Management*, *60*, 102387. https://doi.org/10.1016/j.ijinfomgt.2021.102387

Aldossary, M., Alharbi, H., & Ayub, N. (2024). Optimizing Electric Vehicle (EV) Charging with Integrated Renewable Energy Sources: A Cloud-Based Forecasting Approach for Eco-Sustainability. *Mathematics*, *12*, 2627. https://doi.org/10.3390/math12172627

Alharbi, F., & Csala, D. (2022). A Seasonal Autoregressive Integrated Moving Average with Exogenous Factors (SARIMAX) Forecasting Model-Based Time Series Approach. *Inventions*, *7*, 94. https://doi.org/10.3390/inventions7040094

Bedford, T., Catney, P., & Robinson, Z. (2023). Going down the local: The challenges of place-based net zero governance. *Journal of the British Academy*, *11*, 125–156. https://doi.org/10.5871/jba/011s4.125

Bergmeir, C., & Benítez, J. M. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, *191*, 192–213. https://doi.org/10.1016/j.ins.2011.12.028

Bertsimas, D., Brown, D., & Caramanis, C. (2010). Theory and Applications of Robust Optimization. *SIAM Review*, *53*. https://doi.org/10.1137/080734510

Bhatt, R., Giang, A., Javed, B., & Kandlikar, M. (2024). Equitable charging infrastructure for electric vehicles: Access and experience. *Progress in Energy*, *6*. https://doi.org/10.1088/2516-1083/ad4b8f

Biswal, B., Deb, S., Datta, S., Ustun, T. S., & Cali, U. (2024). Review on smart grid load forecasting for smart energy management using machine learning and deep learning techniques. *Energy Reports*, *12*, 3654–3670. https://doi.org/10.1016/j.egyr.2024.09.056

Bouchlaghem, Y., Akhiat, Y., & Amjad, S. (2022). Feature Selection: A Review and Comparative Study. *E3S Web of Conferences*, *351*, 01046. https://doi.org/10.1051/e3sconf/202235101046

Bracale, A., Caramia, P., De Falco, P., & Hong, T. (2020). Multivariate Quantile Regression for Short-Term Probabilistic Load Forecasting. *IEEE Transactions on Power Systems*, *35*(1), 628–638. https://doi.org/10.1109/TPWRS.2019.2924224

Brinkel, N., van Wijk, T., Buijze, A., Panda, N. K., Meersmans, J., Markotić, P., van der Ree, B., Fidder, H., de Brey, B., Tindemans, S., AlSkaif, T., & van Sark, W. (2024). Enhancing smart charging in electric vehicles by addressing paused and delayed charging problems. *Nature Communications*, *15*, 5089. https://doi.org/10.1038/s41467-024-48477-w

Buolamwini, J., & Gebru, T. (2018). Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, 77–91. https://proceedings.mlr.press/v81/buolamwini18a.html

Cai, H., Xin, Y., Martin, H., & Raubal, M. (2022). Optimizing Electric Vehicle Charging Schedules Based on Probabilistic Forecast of Individual Mobility. *AGILE: GIScience Series*, *3*, 1–13. https://doi.org/10.5194/agile-giss-3-3-2022

Cao, T., Xu, Y., Liu, G., Tao, S., Tang, W., & Sun, H. (2024). Feature-enhanced deep learning method for electric vehicle charging demand probabilistic forecasting of charging station. *Applied Energy*, *371*, 123751. https://doi.org/10.1016/j.apenergy.2024.123751

Chang, M., Bae, S., Cha, G., & Yoo, J. (2021). Aggregated Electric Vehicle Fast-Charging Power Demand Analysis and Forecast Based on LSTM Neural Network. *Sustainability*, *13*, 13783. https://doi.org/10.3390/su132413783

Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System* (p. 794). https://doi.org/10.1145/2939672.2939785

Clement-Nyns, K., Haesen, E., & Driesen, J. (2010). The Impact of Charging Plug-In Hybrid Electric Vehicles on a Residential Distribution Grid. *Power Systems, IEEE Transactions On*, *25*, 371–380. https://doi.org/10.1109/TPWRS.2009.2036481

Committee of Public Accounts. (2024). Public charge points for electric vehicles. *House of Commons*.

Corbett-Davies, S., Gaebler, J. D., Nilforoshan, H., Shroff, R., & Goel, S. (2023). *The Measure and Mismeasure of Fairness* (No. arXiv:1808.00023). arXiv. https://doi.org/10.48550/arXiv.1808.00023

Das, H. S., Rahman, M. M., Li, S., & Tan, C. W. (2020). Electric vehicles standards, charging infrastructure, and impact on grid integration: A technological review. *Renewable and Sustainable Energy Reviews*, *120*, 109618. https://doi.org/10.1016/j.rser.2019.109618

de Amorim, L., Cavalcanti, G., & Cruz, R. (2022). The choice of scaling technique matters for classification performance. *Applied Soft Computing*, *133*, 109924. https://doi.org/10.1016/j.asoc.2022.109924

Deans, A., & Moran, N. (2022). PERTH AND KINROSS COUNCIL. *Head of Planning & Development Report*, *(Report No. 22/191)*.

Dulac-Arnold, G., Mankowitz, D., & Hester, T. (2019). *Challenges of Real-World Reinforcement Learning* (No. arXiv:1904.12901). arXiv. https://doi.org/10.48550/arXiv.1904.12901

Efthymiou, D., Chrysostomou, K., Morfoulaki, M., & Ayfantopoulou, G. (2015). *Electric Vehicle Charging Facility Location Problem: The case of Thessaloniki*.

Ejiyi, C. J., Cai, D., Thomas, D., Obiora, S., Osei-Mensah, E., Acen, C., Eze, F. O., Sam, F., Zhang, Q., & Bamisile, O. O. (2025). Comprehensive review of artificial intelligence

applications in renewable energy systems: Current implementations and emerging trends. *Journal of Big Data*, *12*(1), 169. https://doi.org/10.1186/s40537-025-01178-7

Elahe, M., Kabir, M. A., Mahmud, S. M. H., & Azim, R. (2022). Factors Impacting Short-Term Load Forecasting of Charging Station to Electric Vehicle. *Electronics*, *12*, 55. https://doi.org/10.3390/electronics12010055

Energy Networks Association. (2025). *How do the UK's energy networks work?* Energy Networks Association. https://www.energynetworks.org/energy-networks-explained/

Fan, X., Zhang, X., & Yu, X. B. (2023). Uncertainty quantification of a deep learning model for failure rate prediction of water distribution networks. *Reliability Engineering & System Safety*, *236*, 109088. https://doi.org/10.1016/j.ress.2023.109088

Faridimehr, S., Venkatachalam, S., & Chinnam, R. B. (2019). A Stochastic Programming Approach for Electric Vehicle Charging Network Design. *IEEE Transactions on Intelligent Transportation Systems*, *20*(5), 1870–1882. https://doi.org/10.1109/TITS.2018.2841391

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Gambella, C., Ghaddar, B., & Naoum-Sawaya, J. (2021). Optimization problems for machine learning: A survey. *European Journal of Operational Research*, *290*(3), 807–828. https://doi.org/10.1016/j.ejor.2020.08.045

Geels, F. W. (2019). Socio-technical transitions to sustainability: A review of criticisms and elaborations of the Multi-Level Perspective. *Current Opinion in Environmental Sustainability*, *39*, 187–201. https://doi.org/10.1016/j.cosust.2019.06.009

Gneiting, T., & Katzfuss, M. (2014). Probabilistic Forecasting. *Annual Review of Statistics and Its Application*, *1*(Volume 1, 2014), 125–151. https://doi.org/10.1146/annurev-statistics-062713-085831

GOV.UK. (2023, January 18). *Electric vehicle smart charging action plan*. GOV.UK. https://www.gov.uk/government/publications/electric-vehicle-smart-charging-action-plan/electric-vehicle-smart-charging-action-plan

GOV.UK. (2025). *UK AI sector attracts £200 million a day in private investment since July*. GOV.UK. https://www.gov.uk/government/news/uk-ai-sector-attracts-200-million-a-day-in-private-investment-since-july

Grimes, D., Ifrim, G., O'Sullivan, B., & Simonis, H. (2014). Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. *Sustainable Computing: Informatics and Systems*, *4*(4), 276–291. https://doi.org/10.1016/j.suscom.2014.08.009

Halevy, A., Norvig, P., & Pereira, F. (2009). The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, *24*(2), 8–12. https://doi.org/10.1109/MIS.2009.36

Hong, T., & Fan, S. (2016). Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, *32*(3), 914–938. https://doi.org/10.1016/j.ijforecast.2015.11.011

Hopkins, E., Potoglou, D., Orford, S., & Cipcigan, L. (2023). Can the equitable roll out of electric vehicle charging infrastructure be achieved? *Renewable and Sustainable Energy Reviews*, *182*, 113398. https://doi.org/10.1016/j.rser.2023.113398

Hsu, C., Chang, C., & Lin, C.-J. (2003). *A Practical Guide to Support Vector Classification Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin*.

Ingrams, S. (2025, April 2). *Time of use tariffs explained*. Which? https://www.which.co.uk/reviews/cutting-your-energy-bills/article/time-of-use-tariffs-explained-aSifC2u9LCGa

Ivanovich Vatin, N., & Madhavi, A. (2024). Enhancing electric vehicle efficiency through model predictive control of power electronics. *MATEC Web of Conferences*, *392*, 01168. https://doi.org/10.1051/matecconf/202439201168

Kangalli Uyar, S. G., Ozbay, B. K., & Dal, B. (2025). Interpretable building energy performance prediction using XGBoost Quantile Regression. *Energy and Buildings*, *340*, 115815. https://doi.org/10.1016/j.enbuild.2025.115815

Khan, H. A. U., Price, S., Avraam, C., & Dvorkin, Y. (2022). Inequitable access to EV charging infrastructure. *Electricity Journal*, *35*(3). https://doi.org/10.1016/j.tej.2022.107096

Kipf, T. N., & Welling, M. (2017). *Semi-Supervised Classification with Graph Convolutional Networks* (No. arXiv:1609.02907). arXiv. https://doi.org/10.48550/arXiv.1609.02907

Koohfar, S., Woldemariam, W., & Kumar, A. (2023). Prediction of Electric Vehicles Charging Demand: A Transformer-Based Deep Learning Approach. *Sustainability*, *15*(3), Article 3. https://doi.org/10.3390/su15032105

Krishna, G. H., Babu, K. V. S. M., Dwivedi, D., Chakraborty, P., Yemula, P. K., & Pal, M. (2024). *Energy Sharing among Resources within Electrical Distribution Systems: A Systematic Review* (No. arXiv:2401.01597). arXiv. https://doi.org/10.48550/arXiv.2401.01597

Lai, G., Chang, W.-C., Yang, Y., & Liu, H. (2018). *Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks* (No. arXiv:1703.07015). arXiv. https://doi.org/10.48550/arXiv.1703.07015

Li, Y., Su, T., Zhao, J., & Yang, R. (2024). *EV Forecasting-Based Model Predictive Control for Distribution System Congestion Mitigation*. 2024 IEEE Kansas Power and Energy Conference (KPEC). https://doi.org/10.1109/KPEC61529.2024.10676202

Logan, K. G., Nelson, J. D., Brand, C., & Hastings, A. (2021). Phasing in electric vehicles: Does policy focusing on operating emission achieve net zero emissions reduction objectives? *Transportation Research Part A: Policy and Practice*, *152*, 100–114. https://doi.org/10.1016/j.tra.2021.08.001

Mahajan, M., Kumar, S., Pant, B., & Tiwari, U. K. (2020). Incremental Outlier Detection in Air Quality Data Using Statistical Methods. *2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy (ICDABI)*, 1–5. https://doi.org/10.1109/ICDABI51230.2020.9325683

Mandolakani, F., & Singleton, P. (2024). Electric vehicle charging infrastructure deployment: A discussion of equity and justice theories and accessibility measurement. *Transportation Research Interdisciplinary Perspectives*, *24*, 101072. https://doi.org/10.1016/j.trip.2024.101072

Manigandan, P., Alam, M. S., Alharthi, M., Khan, U., Alagirisamy, K., Pachiyappan, D., & Rehman, A. (2021). Forecasting Natural Gas Production and Consumption in United States-Evidence from SARIMA and SARIMAX Models. *Energies*, *14*(19), 1–17.

Manny, L., Angst, M., Rieckermann, J., & Fischer, M. (2022). Socio-technical networks of infrastructure management: Network concepts and motifs for studying digitalization, decentralization, and integrated management. *Journal of Environmental Management*, *318*, 115596. https://doi.org/10.1016/j.jenvman.2022.115596

Manousakis, N. M., Karagiannopoulos, P. S., Tsekouras, G. J., & Kanellos, F. D. (2023). Integration of Renewable Energy and Electric Vehicles in Power Systems: A Review. *Processes*, *11*(5), 1544. https://doi.org/10.3390/pr11051544

Marzbnai, F., Osman, A., & Hassan, M. (2023). Electric Vehicle Energy Demand Prediction Techniques: An In-Depth and Critical Systematic Review. *IEEE Access*, *PP*, 1–1. https://doi.org/10.1109/ACCESS.2023.3308928

Mayne, D. (2000). Nonlinear Model Predictive Control:Challenges and Opportunities. In F. Allgöwer & A. Zheng (Eds.), *Nonlinear Model Predictive Control* (pp. 23–44). Birkhäuser Basel. https://doi.org/10.1007/978-3-0348-8407-5_2

Mckenna, E., & Thomson, M. (2016). High-resolution stochastic integrated thermal–electrical domestic demand model. *Applied Energy*, *165*, 445–461. https://doi.org/10.1016/j.apenergy.2015.12.089

Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2022). *A Survey on Bias and Fairness in Machine Learning* (No. arXiv:1908.09635). arXiv. https://doi.org/10.48550/arXiv.1908.09635

Mocanu, E., Nguyen, P. H., Gibescu, M., & Kling, W. L. (2016). Deep learning for estimating building energy consumption. *Sustainable Energy, Grids and Networks*, *6*, 91–99. https://doi.org/10.1016/j.segan.2016.02.005

Muratori, M. (2018). Impact of uncoordinated plug-in electric vehicle charging on residential power demand. *Nature Energy*, *3*(3), 193–201. https://doi.org/10.1038/s41560-017-0074-z

Mwasilu, F., Justo, J. J., Kim, E.-K., Do, T., & Jung, J.-W. (2014). Electric vehicles and smart grid interaction: A review on vehicle to grid and renewable energy sources integration. *Renewable and Sustainable Energy Reviews*, *34*, 501–516. https://doi.org/10.1016/j.rser.2014.03.031

National Gried ESO. (2025). *Future Energy Scenarios 2025: Pathways to Net Zero*.

Octopus Energy. (n.d.). *Octopus Power-ups | Free green electricity in NGED*. Octopus Energy. Retrieved 2 August 2025, from https://octopus.energy/power-ups-national-grid-ed/

Ortega-Vazquez, M. A. (2014). Optimal scheduling of electric vehicle charging and vehicle-to-grid services at household level including battery degradation and price uncertainty. *IET Generation, Transmission & Distribution*, *8*(6), 1007–1016. https://doi.org/10.1049/iet-gtd.2013.0624

Osama, M. (2024, August 12). *Study Helps Optimize EV Charging Stations with Integrated PV Forecasting*. AZoCleantech. https://www.azocleantech.com/news.aspx?newsID=35076

Ostermann, A., & Haug, T. (2024). Probabilistic forecast of electric vehicle charging demand: Analysis of different aggregation levels and energy procurement. *Energy Informatics*, *7*(1), 13. https://doi.org/10.1186/s42162-024-00319-1

Pai, P.-F., & Hong, W.-C. (2005). Support vector machines with simulated annealing algorithms in electricity load forecasting. *Energy Conversion and Management*, *46*, 2669–2688. https://doi.org/10.1016/j.enconman.2005.02.004

Pinson, P. (2013). Wind Energy: Forecasting Challenges for Its Operational Management. *Statistical Science*, *28*(4), 564–585. https://doi.org/10.1214/13-STS445

Quiros-Tortos, J., Ochoa, L. F., & Lees, B. (2015). A statistical analysis of EV charging behavior in the UK. *2015 IEEE PES Innovative Smart Grid Technologies Latin America (ISGT LATAM)*, 445–449. https://doi.org/10.1109/ISGT-LA.2015.7381196

Raji, I., Smart, A., White, R., Mitchell, M., Gebru, T., Hutchinson, B., Smith-Loud, J., Theron, D., & Barnes, P. (2020). *Closing the AI accountability gap: Defining an end-to-end*

framework for internal algorithmic auditing (p. 44). https://doi.org/10.1145/3351095.3372873

Ran, J., Gong, Y., Hu, Y., & Cai, J. (2025). EV load forecasting using a refined CNN-LSTM-AM. *Electric Power Systems Research*, *238*, 111091. https://doi.org/10.1016/j.epsr.2024.111091

Rashid, M., Elfouly, T., & Chen, N. (2024). A Comprehensive Survey of Electric Vehicle Charging Demand Forecasting Techniques. *IEEE Open Journal of Vehicular Technology*, *PP*, 1–25. https://doi.org/10.1109/OJVT.2024.3457499

Richardson, P., Flynn, D., & Keane, A. (2012). Optimal Charging of Electric Vehicles in Low-Voltage Distribution Systems. *IEEE Transactions on Power Systems*, *27*(1), 268–279. https://doi.org/10.1109/TPWRS.2011.2158247

Rousseeuw, P., & Leroy, A. (1987). Robust Regression & Outlier Detection, John Wiley & Sons. *Journal of Educational Statistics*, *13*, 358–364.

Rudin, C. (2019). Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. *Nature Machine Intelligence*, *1*, 206–215. https://doi.org/10.1038/s42256-019-0048-x

Shahriar, S., Al-Ali, A., Osman, A., Dhou, S., & Nijim, M. (2021). Prediction of EV Charging Behavior Using Machine Learning. *IEEE Access*, *PP*, 1–1. https://doi.org/10.1109/ACCESS.2021.3103119

Shalev-Shwartz, S., & Ben-David, S. (2013). Understanding Machine Learning: From Theory to Algorithms. *Understanding Machine Learning: From Theory to Algorithms*. https://doi.org/10.1017/CBO9781107298019

Shams, M. H., Niaz, H., Hashemi, B., Jay Liu, J., Siano, P., & Anvari-Moghaddam, A. (2021). Artificial intelligence-based prediction and analysis of the oversupply of wind and solar energy in power systems. *Energy Conversion and Management*, *250*, 114892. https://doi.org/10.1016/j.enconman.2021.114892

Shern, S. J., Sarker, M. T., Haram, M. H. S. M., Ramasamy, G., Thiagarajah, S. P., & Al Farid, F. (2024). Artificial Intelligence Optimization for User Prediction and Efficient Energy

Distribution in Electric Vehicle Smart Charging Systems. *Energies*, *17*(22), 5772. https://doi.org/10.3390/en17225772

Siddique, T., Mahmud, M. S., Keesee, A. M., Ngwira, C. M., & Connor, H. (2022). A Survey of Uncertainty Quantification in Machine Learning for Space Weather Prediction. *Geosciences*, *12*(1), Article 1. https://doi.org/10.3390/geosciences12010027

Singh, A. R., Vishnuram, P., Alagarsamy, S., Bajaj, M., Blazek, V., Damaj, I., Rathore, R. S., Al-Wesabi, F. N., & Othman, K. M. (2024). Electric vehicle charging technologies, infrastructure expansion, grid integration strategies, and their role in promoting sustainable e-mobility. *Alexandria Engineering Journal*, *105*, 300–330. https://doi.org/10.1016/j.aej.2024.06.093

Soroudi, A., & Keane, A. (2014). Robust optimization based EV charging. *2014 IEEE International Electric Vehicle Conference (IEVC)*, 1–6. https://doi.org/10.1109/IEVC.2014.7056223

Sovacool, B. K., Hess, D. J., & Cantoni, R. (2021). Energy transitions from the cradle to the grave: A *meta*-theoretical framework integrating responsible innovation, social practices, and energy justice. *Energy Research & Social Science*, *75*, 102027. https://doi.org/10.1016/j.erss.2021.102027

Steinhilber, S., Wells, P., & Thankappan, S. (2013). Socio-technical inertia: Understanding the barriers to electric vehicles. *Energy Policy*. https://doi.org/10.1016/j.enpol.2013.04.076

The Public Charge Point Regulations. (2024, October). *Public Charge Point Regulations 2023 guidance*. GOV.UK. https://www.gov.uk/government/publications/the-public-charge-point-regulations-2023-guidance/public-charge-point-regulations-2023-guidance

Trist, E. L., & Bamforth, K. W. (1951). Some Social and Psychological Consequences of the Longwall Method of Coal-Getting: An Examination of the Psychological Situation and Defences of a Work Group in Relation to the Social Structure and Technological Content of the Work System. *Human Relations*, *4*(1), 3–38. https://doi.org/10.1177/001872675100400101

Trust, E. S. (2019, October 22). Successful Switched on Towns and Cities announced. *Energy Saving Trust*. https://energysavingtrust.org.uk/successful-switched-towns-and-cities-announced/

Ullah, I., Liu, K., Yamamoto, T., Al Mamlook, R. E., & Jamal, A. (2022). A comparative performance of machine learning algorithm to predict electric vehicles energy consumption: A path towards sustainability. *Energy & Environment*, *33*(8), 1583–1612. https://doi.org/10.1177/0958305X211044998

Uralde, J., Barambones, O., del Rio, A., Calvo, I., & Artetxe, E. (2024). Rule-Based Operation Mode Control Strategy for the Energy Management of a Fuel Cell Electric Vehicle. *Batteries*, *10*(6), Article 6. https://doi.org/10.3390/batteries10060214

Van Kriekinge, G., De Cauwer, C., Sapountzoglou, N., Coosemans, T., & Messagie, M. (2021). Day-Ahead Forecast of Electric Vehicle Charging Demand with Deep Neural Networks. *World Electric Vehicle Journal*, *12*(4), Article 4. https://doi.org/10.3390/wevj12040178

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). *Attention Is All You Need* (No. arXiv:1706.03762). arXiv. https://doi.org/10.48550/arXiv.1706.03762

Vázquez-Canteli, J. R., & Nagy, Z. (2019). Reinforcement learning for demand response: A review of algorithms and modeling techniques. *Applied Energy*, *235*, 1072–1089. https://doi.org/10.1016/j.apenergy.2018.11.002

Vergara, D., Hernández, S., Valdenegro-Toro, M., & Jorquera, F. (2019). *Improving Predictive Uncertainty Estimation using Dropout—Hamiltonian Monte Carlo* (No. arXiv:1805.04756). arXiv. https://doi.org/10.48550/arXiv.1805.04756

Vishnu, G., Kaliyaperumal, D., Pati, P. B., Karthick, A., Subbanna, N., & Ghosh, A. (2023). Short-Term Forecasting of Electric Vehicle Load Using Time Series, Machine Learning, and Deep Learning Techniques. *World Electric Vehicle Journal*, *14*, 266. https://doi.org/10.3390/wevj14090266

Xu, P., Wang, X., & Li, Z. (2025). Impact and optimization of vehicle charging scheduling on regional clean energy power supply network management. *Energy Informatics*, *8*(1), 13. https://doi.org/10.1186/s42162-025-00476-x

Yang, Y., Yeh, H.-G., & Nguyen, R. (2022). A Robust Model Predictive Control-Based Scheduling Approach for Electric Vehicle Charging With Photovoltaic Systems. *IEEE Systems Journal*, *PP*, 1–11. https://doi.org/10.1109/JSYST.2022.3183626

Yin, X., Fallah Shorshani, M., McConnell, R., Fruin, S., & Chiang, Y.-Y. (2023). *Quantile Extreme Gradient Boosting for Uncertainty Quantification*. https://doi.org/10.48550/arXiv.2304.11732

Zhang, Y., Liu, C., Rao, X., Zhang, X., & Zhou, Y. (2023). Spatial-temporal load forecasting of electric vehicle charging stations based on graph neural network. *Journal of Intelligent & Fuzzy Systems*, *46*, 1–16. https://doi.org/10.3233/JIFS-231775

Zheng, J., Cui, J., Zhao, Z., Li, G., Wang, C., Lu, Z., Yang, X., & Liu, Z. (2025). Multi-Objective Optimization Scheduling for Electric Vehicle Charging and Discharging: Peak-Load Shifting Strategy Based on Monte Carlo Sampling. *Designs*, *9*(2), Article 2. https://doi.org/10.3390/designs9020051

Zhou, Z., Jiang, B., & Wang, Q. (2025). *Mixer-Informer-Based Two-Stage Transfer Learning for Long-Sequence Load Forecasting in Newly Constructed Electric Vehicle Charging Stations* (No. arXiv:2505.06657). arXiv. https://doi.org/10.48550/arXiv.2505.06657

Zhou, Z., Liu, C., & Botterud, A. (2016). *Stochastic Methods Applied to Power System Operations with Renewable Energy: A Review* (No. ANL/ESD--16/14). Argonne National Lab. (ANL), Argonne, IL (United States). https://doi.org/10.2172/1307655

Appendix A: Data pre-processing and feature engineering

This appendix explains what data was used, how each field is defined, how the data was cleaned, and where to find the raw dataset. It is here so another reader can follow every step.

Session dataset (2016–2019): field definitions and cleaning log

Table 5 lists the core fields for each charging session: identifiers, hardware details, start and end timestamps, energy, and site name. Times are recorded in local time (Europe/London).

Table 5: PKC session dataset (2016–2019): field definitions

Field	Description
_id	A unique identifier for the database record.
CP_ID	A unique identifier for the specific charge-point hardware.
Connector	The type of physical connector used for the session.
Start_Date & Start_Time	The date and time for the beginning of each charging session.
End_Date & End_Time	The date and time for the end of each charging session.
Total_kWh	The total energy consumed during the session is measured in kilowatt-hours.
Site	The name of the charging location (e.g., a specific car park).
Model	The model of the charging station hardware.

Public station inventory (2023): field definitions

Table 6 describes the fields for the recent PKC public network, including British National Grid coordinates (EPSG:27700), charger speed, connector types, payment model, site name, and address.

Table 6: PKC Public Stations (2023): Field Definitions.

Field	Description
Geographic Coordinates	Latitude and longitude (x and y coordinates in the British National Grid format).
Charger Specifications	Details on the charger speed (CH_SPEED), connector types (CON_TYPE), and payment model (CP_CHARGE - e.g., Free or Charge).
Site Information	The name and address of the charging location (CP_Name, ADDRESS).

Final forecasting inputs and encodings

Table 7 lists the inputs used by the forecasting model. They are grouped as:

- **temporal:** hour of day, day-of-week indicators, season indicators
- **short-term history:** Previous 24HrAverageDemand
- **cross-site context:** HourlyAverageDemand
- **power proxy:** Pavg_kW, the target is Target_Hourly_kWh (aggregated hourly energy to be predicted). One-hot encodings are used to ensure the final feature dimensions are clear.

Table 7: Final forecasting inputs and encodings.

Feature	Type	Description
Pavg_kW	Numeric	Average power demand (kW) during the connection hour
ConnectionHour	Numeric	Hour of day (0–23) when the charging session began.
WorkingStatus	Binary	Indicates weekday (1) or weekend (0).
HourlyAverageDemand	Numeric	Mean demand (kWh) over all sites in the same hour, averaged across the dataset.
Previous24HrAverageDemand	Numeric	Mean demand (kWh) over the preceding 24 hours, capturing short-term trends.

Season	One-hot categorical	Encoded as three binary columns – Spring, Summer, Winter – to capture seasonal effects on charging behaviour. One-hot encoding creates a separate "1/0" column per season.
Day of Week	One-hot categorical	Encoded as seven binary columns – Mon, Tue, Wed, Thu, Fri, Sat, Sun – to model weekday/weekend and day-specific patterns. One-hot encoding ensures each day is mutually exclusive.
Target: Total_kWh	Numeric	Actual energy consumed (kWh) during the charging session – this is the variable the model aims to forecast.

Data cleaning quantified

Table 8 shows record counts at each cleaning step. It reports the number of rows remaining after removing missing values, invalid sessions (where Total_kWh ≤ 0), and outliers. The final row gives the total kept and the total removed. This makes the impact of each filter visible immediately.

Table 8: Record counts at each data cleaning step.

Stage	**Number of Records**	**Records Removed**
Initial Dataset	55,878	-
After removing missing values (Before hourly aggregation)	54,335	1543
After removing invalid sessions before hourly aggregation (Total_kWh <= 0)	54,101	234
After outlier removal before hourly aggregation (IQR method)	52,227	1829
Final Records for Analysis (After hourly aggregation)	26,057	-

Data availability and link to the main dataset

The charging sessions datasets can be accessed here: PKC EV charging stations 2016 to 2019, while the current charging stations dataset can be accessed here: PKC charging stations 2023 (Most recently accessed 18 July 2025). The licensing information of both datasets can be found at Dataset License (Most recently accessed 18 July 2025).

How this ties to the main text: Section 3 gives a short overview of inputs. Appendix A holds the full field lists, the final set of features, and a simple audit of how many rows were removed at each step. That makes the workflow easy to check and reproduce.

Appendix B: Forecasting Model Selection and Hyperparameter Tuning

This appendix demonstrates the methodological rigour behind the model selection process, justifying the choice of XGBoost and detailing its optimisation.

Comparative Performance of Forecasting Models

To select the most effective forecasting method, a range of models representing different algorithmic classes were trained and evaluated. The performance metrics in Table 9 show that while all ensemble methods performed strongly, the optimised XGBoost model delivered the best overall performance, confirming its suitability for the final forecasting task.

Table 9: Comparative Model Performance on Test Set.

Model Class	Model	MAE (kWh)	RMSE (kWh)	R^2
Ensemble Learning	Optimised XGBoost	15.83	26.59	0.64
Ensemble Learning	Optimised Random Forest	15.95	27.32	0.61
Ensemble Learning	Optimised LightGBM	17.13	28.37	0.58
Deep Learning	Tuned CNN-LSTM	18.43	28.97	0.56
Linear Model	OLS	24.05	31.29	0.49
Kernel-Based ML	SVR	21.55	36.44	0.31
Statistical Time Series	SARIMA	24.01	36.79	0.29

XGBoost Hyperparameter Tuning Protocol

The final XGBoost model was optimised using the `Optuna` library. The process involved running 50 trials to find the hyperparameter combination that minimised the Root Mean Squared Error (RMSE) on the validation set. Table 10 shows the final parameters, identified by the Optuna study, which were used to train the final model.

Table 10: Final Optimised Hyperparameters for XGBoost.

Hyperparameter	**Final Value**
n_estimators	530
learning_rate	0.01
max_depth	7
subsample	0.90
colsample_bytree	0.67
min_child_weight	6
gamma	0.66
lambda (L2)	0.01
alpha (L1)	$4.6e^{-05}$

Appendix C: Sensitivity Analysis of Optimisation Results

This appendix addresses a limitation of the study by testing the robustness of the optimisation findings under different conditions. A series of sensitivity analyses was performed to evaluate how the total cost savings from the forecast-informed strategy change when key parameters are adjusted.

Impact of Varying Demand Charge

This analysis examines how savings are impacted by changes in the demand charge, a crucial component of commercial electricity tariffs. The optimisation was run using a lower (£10/kW), baseline (£12/kW), and higher (£15/kW) demand charge. The result is shown in Table 11.

Table 11: Sensitivity of Optimisation Results to Varying Demand Charges.

Demand Charge (£/kW)	Total Optimised Cost (£)	Total Savings (%)
£10.00	£762.55	59.2%
£12.00	£874.64	53.2%
£15.00	£1,042.77	44.2%

Sensitivity to Forecast Uncertainty Cap

This analysis examines how altering the "safety margin" of the forecast affects the cost and feasibility of the optimised schedule. It compares the results when using the 90th percentile forecast as the upper power limit versus a more conservative 95th percentile. The results in Table 12 show that increasing the cap to the 95th percentile results in a higher operational cost, as the optimiser reserves more capacity to handle potential demand surges.

Table 12: Sensitivity of Optimisation Results to the Forecast Uncertainty Cap.

Uncertainty Cap	Total Optimised Cost (£)	Total Savings (%)
90th Percentile	£787.34	57.9%
95th Percentile	£950.40	49.1%

Sensitivity Optimised Model TOU Price Differential

This analysis tests how the financial incentive – the price difference between peak and off-peak electricity – affects the savings generated by the optimiser. To do this, three distinct Time-of-Use (TOU) price profiles were created to represent different levels of financial incentive:

- A **Low Differential** profile with a smaller price gap between peak and off-peak hours.
- A **Baseline Differential** profile representing the standard tariff used in the main study.
- A **High Differential** profile with a larger, more punitive gap between peak and off-peak prices.

For each of these three price scenarios, a benchmark cost was established using the "Uncontrolled Charging" baseline. This baseline represents the actual, historical charging demand from the dataset, simulating a "do-nothing" approach where users charge based on convenience. The forecast-informed optimiser was then run against the same tariff.

The results in Table 13 compare the optimised cost to this uncontrolled baseline for each scenario. They show that the optimiser delivers remarkably stable savings of around 57–58%, indicating that its value is robust even with varying energy price incentives, largely due to its effectiveness at mitigating the consistent peak demand charge.

Table 13: Sensitivity of Optimisation Results to Varying TOU Price Differentials.

TOU Profile	Total Optimised Cost (£)	Total Savings (%)
Low Differential	£778.53	58.1%
Baseline Differential	£787.34	57.9%
High Differential	£806.34	57.4%

Sensitivity of Rule-Based Charging to TOU Price Differential

To provide a comprehensive comparison, a sensitivity analysis was also performed on the simpler "Rule-Based Charging" strategy. This analysis tested how a non-predictive strategy, which schedules all charging to occur only during the cheapest tariff hours, performs under the different TOU profiles.

The results in Table 14 show that this simple strategy is highly ineffective and can even be detrimental. In the Low and Baseline differential scenarios, the strategy increased the total cost compared to doing nothing, resulting in negative savings. This is because concentrating the

entire day's energy demand into a few hours creates an extremely high-power peak, leading to punitive demand charges that overwhelm any savings made on energy costs.

Table 14: Sensitivity of the Rule-Based Strategy to Varying TOU Prices.

TOU Profile	**Total Rule-Based Cost (£)**	**Total Savings (%)**
Low Differential	£778.53	-3.5%
Baseline Differential	£787.34	-1.4%
High Differential	£806.34	0.7%

Appendix D: Probabilistic Forecast Calibration

This appendix provides a quantitative and visual assessment of the reliability of the probabilistic forecasts used in this study. It validates the quality of the uncertainty estimates that underpin the optimisation framework.

Calibration Metrics

To evaluate the quality of the uncertainty estimates, two key metrics were calculated for the test set. Coverage measures the percentage of actual observations that fall within the 10^{th}–90^{th} percentile prediction interval (the target is 80%). Pinball Loss is a metric that evaluates the accuracy of quantile forecasts; a lower score is better.

The results in Table 15 show that the model is well-calibrated. The empirical coverage is very close to the target, and the low pinball loss score indicates high accuracy across the quantiles.

Table 15: Forecast Calibration Metrics.

Metric	Target Value	Actual Value
80% Interval Coverage	80.0%	80.4%
Average Pinball Loss	Lower is better	8.9

Reliability Curve

The reliability curve in Figure 26 visually assesses the calibration. A perfectly calibrated model would have its point lie on the diagonal dashed line. The plot shows that the model's empirical coverage is extremely close to the nominal coverage target, indicating a highly reliable and well-calibrated probabilistic forecast. The plot shows the nominal coverage (target of 80%) against the empirical coverage (80.4%) achieved on the test set.

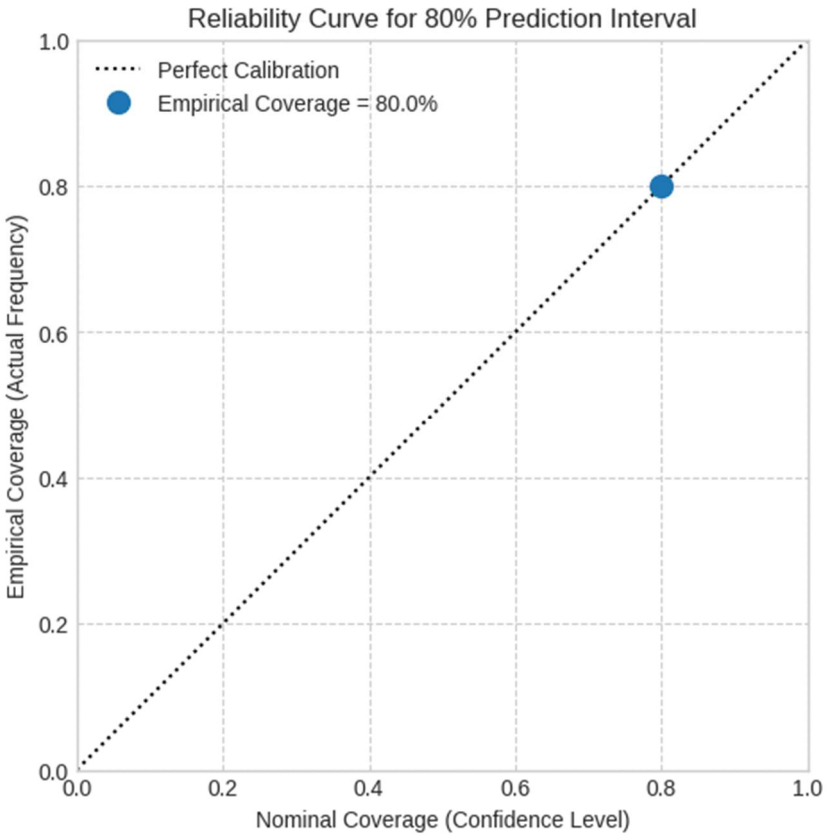


Figure 26: Reliability curve for the 80% prediction interval.

Appendix E: Full Implementation Code

To ensure full transparency and replicability, this appendix provides the complete Python script used for the data processing, hyperparameter tuning, forecasting, and optimisation pipeline described in this dissertation.

Environment Details (`requirements.txt`)

The analysis was conducted using the following primary Python libraries (Figure 27). To replicate the environment, these packages can be installed via pip.

```
pandas==2.1.0
numpy==1.25.2
scikit-learn==1.3.0
xgboost==2.0.0
lightgbm==4.1.0
optuna==3.4.0
pulp==2.7.0
matplotlib==3.7.2
seaborn==0.12.2
```

Figure 27: The Python libraries used for this study.

Full Python Script

The complete, commented Python script is attached to this document.

Appendix F: Optimisation Model Mathematical Formulation

Sets and Indices:

- T: The set of time intervals in the optimisation horizon, where $t \in T = \{1,2,\ldots, H\}$. For this study, $H = 24$ for daily optimisation.

Parameters:

- E_t^{med}: The median forecasted energy demand (kWh) for interval t.
- P_t^{max}: The maximum allowable power (kW) in interval t, derived from the 90th percentile forecast.
- C_t^{en}: The Time-of-Use (TOU) energy price (£/kWh) for interval t.
- C^{dem}: The demand charge price (£/kW) for the peak power over the entire horizon.
- R^{max}: The maximum allowable ramp rate (kW/hour); this is the fastest rate at which the charger's power can increase or decrease over time (Clement-Nyns et al., 2010).

Decision Variables:

- P_t^{sch}: The scheduled charging power (kW) to be dispatched in interval t. This is the primary output of the model.
- P^{peak}: A single variable representing the peak power (kW) over the entire horizon T.

Objective Function: The objective is to minimise the sum of the total energy cost and the demand charge cost.

$$Minimise: Z = \sum_{t=1}^{H}(P_t^{sch} \cdot C_t^{en}) + (P^{peak} \cdot C^{dem}) \quad (22)$$

Constraints: The optimisation is subject to the following constraints:

1. **Energy Adequacy Constraint:** The total scheduled energy must meet or exceed the total expected energy demand from the median forecast.

$$\sum_{t=1}^{H} P_t^{sch} \geq \sum_{t=1}^{H} E_t^{med} \quad (23)$$

2. **Hourly Power Limit (Uncertainty Constraint):** The scheduled power in any hour cannot exceed the robust upper bound from the 90th percentile forecast.

$$P_t^{sch} \leq P_t^{max} \; \forall t \in T \quad (24)$$

3. **Peak Power Constraint:** The peak power variable must be greater than or equal to the scheduled power in every interval.

$$P_t^{peak} \leq P_t^{sch} \; \forall t \in T \tag{25}$$

4. **Ramp Rate Constraints:** The change in scheduled power between consecutive intervals is limited.

$$P_t^{sch} - P_{t-1}^{sc} \leq R^{max} \; \forall t \in \{2, \ldots, H\} \tag{26}$$

$$P_{t-1}^{sc} - P_t^{sch} \leq R^{max} \; \forall t \in \{2, \ldots, H\} \tag{27}$$

Appendix G: Additional Scenario-Based Forecasts and Optimisation Visuals

This appendix provides a detailed visual analysis of the model's performance and the resulting optimisation outcomes across a range of atypical scenarios. These plots showcase the model's ability to generate reliable uncertainty bounds and demonstrate how the forecast-informed strategy adapts to different real-world demand profiles, such as weekends and public holidays.

Forecast with Uncertainty Bands for a Weekend

This plot (Figure E1) shows the median forecast (red) and the 10^{th}–90^{th} percentile uncertainty bands (shaded blue). The bands successfully contain the volatile actual demand (black), demonstrating the probabilistic model's ability to provide a reliable "safety margin" for operational planning, even during unpredictable weekend peaks.

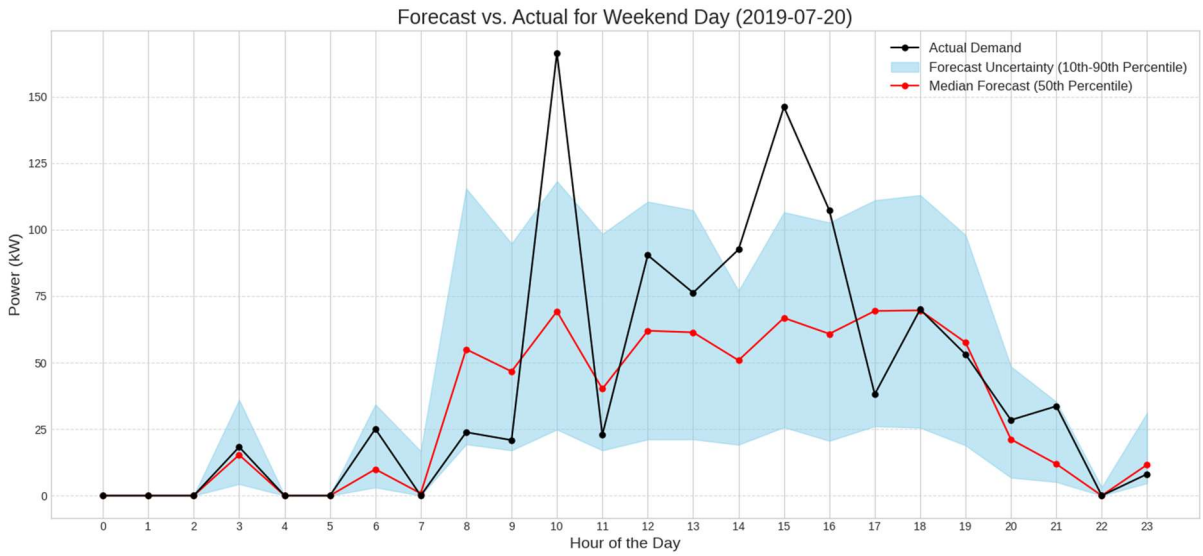


Figure 28: Forecast vs Actual for a weekend day.

Forecast with Uncertainty Bands for a Bank Holiday

Like the weekend plot (Figure E2), this visualisation confirms the reliability of the uncertainty forecast during an atypical day. The prediction interval successfully captures the sharp and irregular peaks of the holiday, providing the robust upper and lower bounds necessary for the downstream optimisation algorithm.

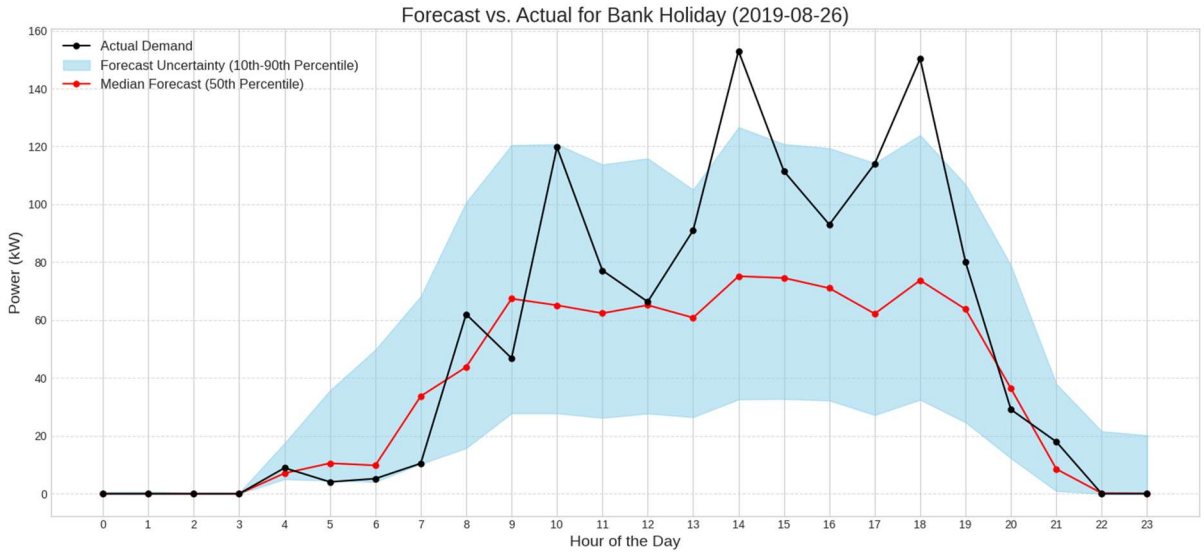


Figure 29: Forecast vs Actual for a Bank Holiday.

Forecast Performance on a Bank Holiday

This plot (Figure E3) compares the Actual Demand (black line) against the Optimised XGBoost Prediction (red line). The visualisation demonstrates the model's ability to handle the significant volatility and sharp, irregular peaks characteristic of a public holiday, tracking the actual demand much more closely than a simpler model would.

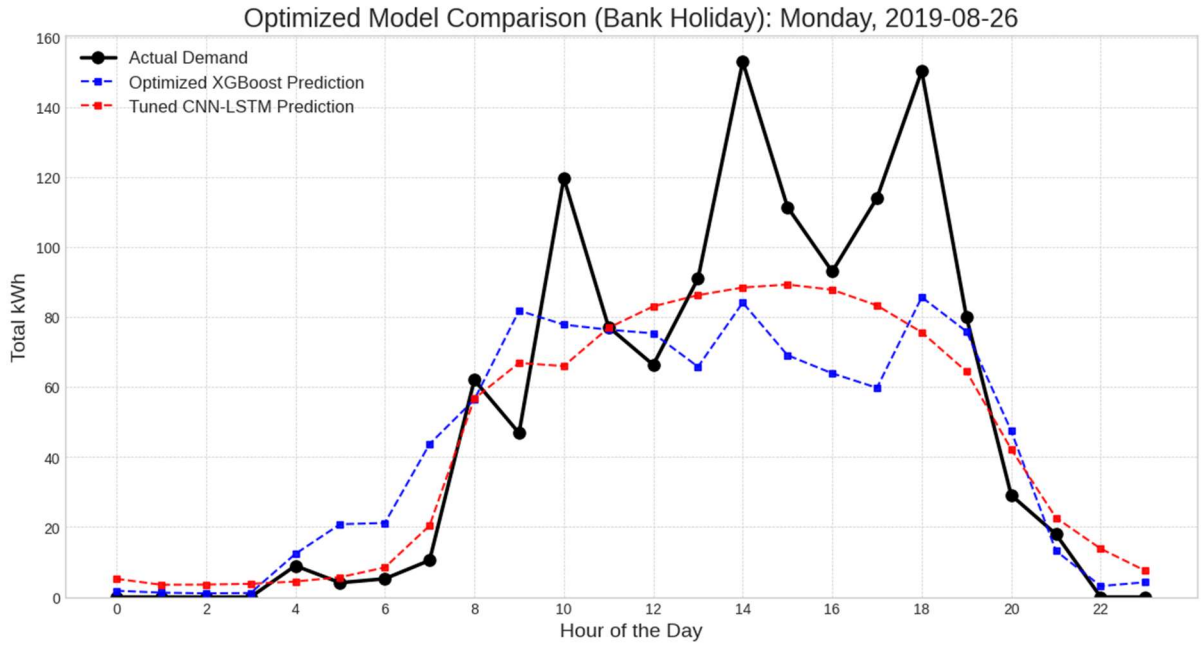


Figure 30: Model performance on a bank holiday.

Forecast Performance During Christmas Week

This plot (Figure E4) serves as a stress test, comparing the Actual Demand (black line) against the XGBoost Forecast (blue line) over a highly irregular holiday period. The forecast successfully navigates the atypical demand patterns across the week, demonstrating the model's robustness and ability to generalise beyond standard weekly routines.

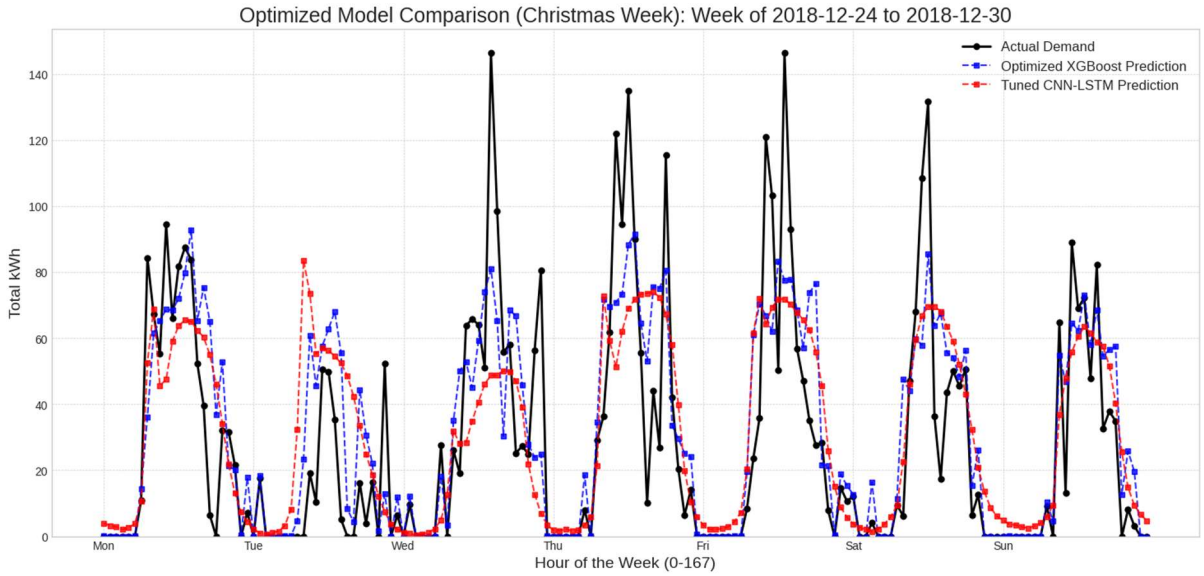


Figure 31: Model performance during the 2018 Christmas Week

Abbreviations

Artificial intelligence
AI

Autoregressive
AR

CHArge de MOve
CHAdeMO

Coefficient of Determination
R^2

Combined Charging System
CCS

Convolutional Neural Network and Long Short-Term Memory
CNN-LSTM

Deep learning
DL

Electric Vehicles
EV

Extreme Gradient Boosting
XGBoost

Graph Neural Networks
GNN

Interquartile Range
IQR

Linear Programming
LP

Local Electric Vehicle Infrastructure
LEVI

Machine learning
ML

Mean Absolute Error
MAE

Mixed-integer Linear Programming
MILP

Model Predictive Control
MPC

Moving Average
MA

Multiple Linear Regression
MLR

Ordinary Least Squares
OLS

Perth and Kinross Council
PKC

Radial Basis Function
RBF

Rectified Linear Unit
ReLU

Recurrent Neural Network
RNN

Root Mean Squared Error
RMSE

Seasonal Autoregressive Integrated Moving Average
SARIMA

Socio-Technical Uncertainty-Aware Optimisation Framework
ST-UOF

Support Vector Regression
SVR

Time-of-Use
TOU

Appendix E: Full Implementation Code

Project Title: A Socio-Technical Framework for Uncertainty-Aware, Forecast-Informed Optimisation of the Public EV Charging Network.

Project Description: This appendix contains the complete Python script used to implement the Socio-Technical Uncertainty-Aware Optimisation Framework (ST-UOF) detailed in this dissertation. The code is structured to follow the methodological pipeline, beginning with data loading and pre-processing, followed by the training and evaluation of machine learning forecasting models, forecast-informed linear programming optimisation and concluding with the social technical analysis of the current infrastructures.

The primary goal of this codebase is to translate historical EV charging data from Perth and Kinross Council into a cost-optimal, 24-hour charging schedule that respects real-world operational constraints. Key libraries used include Pandas for data manipulation, Scikit-learn and XGBoost for machine learning, Optuna for hyperparameter tuning, and PuLP for the optimisation.

This Python notebook implements the end-to-end data science pipeline for the dissertation, covering data cleaning, feature engineering, the training and evaluation of multiple forecasting models, and the final forecast-informed optimisation simulation.

Mount Google Drive

This initial step connects my Google Colab notebook to the user's personal Google Drive, allowing the script to access datasets or other necessary files stored there.

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Description:

This is a recommended procedure for working with files in a Colab environment. The first line imports the necessary drive module from the google.colab library. The drive.mount() function then executes the mounting process. When run, it prompts the user with a URL to authorize access. Once the authorization code is provided, the user's Google Drive becomes accessible within the Colab environment at the path /content/drive.

Setting Up the Tools

This block prepares my coding environment.

Importing Libraries: I am loading the essential toolkits for the project.

- pandas: For handling my data in tables.
- numpy: For numerical calculations.
- matplotlib.pyplot: For creating graphs and charts.

Hiding Warnings: The warnings.filterwarnings('ignore') line keeps my output clean by hiding non-critical warning messages.

Improving Display: The pd.set_option lines make sure that when I view my data tables, they are displayed clearly without being cut off.

In [3]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings

# Suppress warnings for cleaner output
warnings.filterwarnings('ignore')

# Set pandas display options for easier inspection
pd.set_option('display.max_columns', 50)
pd.set_option('display.width', 120)
```

Loading My Data

In this step, I'm loading my dataset into the programming environment.

- df = pd.read_csv(...): I *italicized text*'m using the pandas library which I imported earlier, to read my charging data from a CSV file. A CSV is just a simple text file that stores data in a table format, like a spreadsheet.
- 'EV_Charge_Sessions_2016-2019_perth.csv': This is the name of my data file. I'm telling the program to find this specific file.
- encoding='latin1': Sometimes, data files have special characters that the computer doesn't recognize by default. I've added this part to make sure all the characters in my file are read correctly, preventing any errors during the import.

- df: After loading the data, I'm storing it in a variable named df. Think of df as a container, which now holds my entire dataset as a neatly organized table called a DataFrame. From now on, whenever I want to work with my data, I'll use this df variable.

In [53]:
```
# Load Data and Initial Inspection
csv_path = "/content/drive/MyDrive/CIM/Dissertation/Data/PKC_EVChargeStationUse_

# Read CSV into a DataFrame
df = pd.read_csv(csv_path)

# Initial inspection
print("Shape of dataset:", df.shape)
print("\nFirst five rows:")
df.head()
```

```
Shape of dataset: (55878, 10)

First five rows:
```

Out[53]:

	ID	CP ID	Connector	Start Date	Start Time	End Date	End Time	Total kWh	Site	Model
0	1	51516	2	2019-08-31 0:00:00	23:26	2019-08-31 0:00:00	23:54	12.780	Kinross Park and Ride, Kinross	APT Triple Rapid Charger
1	2	51249	1	2019-08-31 0:00:00	21:12	2019-08-31 0:00:00	21:32	5.700	Atholl Street Car Park, Dunkeld	Siemens Triple Rapid Charger
2	3	50995	2	2019-08-31 0:00:00	20:26	2019-08-31 0:00:00	21:20	27.680	Moness Terrace Car Park, Aberfeldy	APT Triple Rapid Charger
3	4	50839	1	2019-08-31 0:00:00	20:06	2019-08-31 0:00:00	23:03	17.526	Mill Street, Perth	APT 22kW Raption
4	5	50745	2	2019-08-31 0:00:00	19:43	2019-08-31 0:00:00	20:23	12.930	Kinross Park and Ride, Kinross	APT Triple Rapid Charger

Initial Data Overview

I am performing a quick health check on the dataset. I display the column names to see the variables I'm working with, check data types and for missing values, and generate summary statistics to understand the basic distribution of my numerical data.

In [54]:
```
# Initial inspection
print("\nColumn names:", df.columns.tolist())
print("\nData types and missing values:")
```

```
df.info()
print("\nSummary statistics:")
df.describe()
```

```
Column names: ['ID', 'CP ID', 'Connector', 'Start Date', 'Start Time', 'End Dat
e', 'End Time', 'Total kWh', 'Site', 'Model']

Data types and missing values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55878 entries, 0 to 55877
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   ID          55878 non-null  int64
 1   CP ID       55878 non-null  int64
 2   Connector   55878 non-null  int64
 3   Start Date  55878 non-null  object
 4   Start Time  55878 non-null  object
 5   End Date    55716 non-null  object
 6   End Time    55716 non-null  object
 7   Total kWh   55716 non-null  float64
 8   Site        55878 non-null  object
 9   Model       55878 non-null  object
dtypes: float64(1), int64(3), object(6)
memory usage: 4.3+ MB

Summary statistics:
```

Out[54]:

	ID	CP ID	Connector	Total kWh
count	55878.000000	55878.000000	55878.000000	55716.000000
mean	27939.500000	50836.745320	1.732614	10.741379
std	16130.733508	467.415991	0.548847	10.516036
min	1.000000	50244.000000	1.000000	-1394.711000
25%	13970.250000	50286.000000	1.000000	5.030000
50%	27939.500000	50838.000000	2.000000	8.702000
75%	41908.750000	51250.000000	2.000000	14.850000
max	55878.000000	51519.000000	3.000000	138.250000

Data Cleaning and Preprocessing

Here, I am refining the dataset to ensure its quality and consistency for analysis. I first standardize all column names by removing spaces and special characters, which makes them easier to work with. I then remove any rows that are missing critical information, specifically dates, times, or the total energy delivered. I also filter out any records showing non-positive energy consumption, as these represent invalid charging sessions. Finally, I reset the dataframe's index to ensure clean, sequential numbering after the row deletions and confirm the new size of the dataset.

In [55]:
```
# Data Cleaning and Preprocessing
```

```
# Standardise column names for easier access
df.columns = [col.strip().replace(' ', '_').replace('/', '_') for col in df.colu

# Check missing values again, now with new names
print("Missing values per column:\n", df.isnull().sum())

# Remove rows where critical columns are missing (dates, times, kWh)
critical_cols = ['Start_Date', 'Start_Time', 'End_Date', 'End_Time', 'Total_kWh'
df = df.dropna(subset=critical_cols)

# Remove rows with non-positive energy delivered (Total_kWh <= 0)
df = df[df['Total_kWh'] > 0]

# Reset index after dropping rows
df = df.reset_index(drop=True)

print(f"After cleaning, dataset shape: {df.shape}")
```

```
Missing values per column:
 ID              0
CP_ID            0
Connector        0
Start_Date       0
Start_Time       0
End_Date       162
End_Time       162
Total_kWh      162
Site             0
Model            0
dtype: int64
After cleaning, dataset shape: (54335, 10)
```

Quick check to confirm the new shape of my data after dropping missing values.

In [56]:
```
# Confirm nomissing values
print("Missing values per column:\n", df.isnull().sum())
df.describe()
```

```
Missing values per column:
 ID             0
CP_ID           0
Connector       0
Start_Date      0
Start_Time      0
End_Date        0
End_Time        0
Total_kWh       0
Site            0
Model           0
dtype: int64
```

Out[56]:

	ID	CP_ID	Connector	Total_kWh
count	54335.000000	54335.000000	54335.000000	54335.000000
mean	27883.543370	50836.685046	1.743425	11.040054
std	16106.262365	466.088597	0.545633	8.603690
min	1.000000	50244.000000	1.000000	0.001000
25%	13919.500000	50286.000000	1.000000	5.250000
50%	27899.000000	50838.000000	2.000000	8.950000
75%	41814.500000	51250.000000	2.000000	15.010000
max	55878.000000	51519.000000	3.000000	138.250000

Calculating Charging Duration

I am converting the separate date and time columns into a unified datetime format for both the start and end of each charging session. This allows me to calculate the precise duration of every session, which I then convert into hours for straightforward analysis. I also clean the data by removing any records where the start or end times could not be properly parsed or where the calculated duration is invalid (i.e., zero or negative). Finally, I display the new shape of my dataset and the first few rows of the new time-related columns to verify the changes.

In [57]:
```
# Parse date and time columns and calculate charging session duration

# Combine and convert to datetime
df['StartDateTime'] = pd.to_datetime(df['Start_Date'] + ' ' + df['Start_Time'],
df['EndDateTime']   = pd.to_datetime(df['End_Date'] + ' ' + df['End_Time'], dayf

# Remove rows where parsing failed (if any)
df = df.dropna(subset=['StartDateTime', 'EndDateTime'])

# Compute duration in hours
df['ChargingDurationHrs'] = (df['EndDateTime'] - df['StartDateTime']).dt.total_s

# Remove rows with zero or negative durations (invalid sessions)
df = df[df['ChargingDurationHrs'] > 0]

print(f"After parsing datetimes and computing duration, shape: {df.shape}")
print(df[['StartDateTime', 'EndDateTime', 'ChargingDurationHrs']].head())
```

```
After parsing datetimes and computing duration, shape: (54101, 13)
        StartDateTime         EndDateTime  ChargingDurationHrs
0 2019-08-31 23:26:00 2019-08-31 23:54:00             0.466667
1 2019-08-31 21:12:00 2019-08-31 21:32:00             0.333333
2 2019-08-31 20:26:00 2019-08-31 21:20:00             0.900000
3 2019-08-31 20:06:00 2019-08-31 23:03:00             2.950000
4 2019-08-31 19:43:00 2019-08-31 20:23:00             0.666667
```

Aggregating Data to an Hourly Level

I am now transforming the data from individual charging sessions into a format suitable for time-series analysis. First, I calculate the average power (in kW) for each session by dividing the total energy by the duration. After cleaning up any potential calculation errors, I set the session's start time as the main index for my dataset. This allows me to resample the data into fixed hourly intervals. For each hour, I calculate the average power demand and the sum of energy delivered, which gives me a clear, aggregated view of the network's usage over time. Finally, I display the first few rows of this new hourly data to verify the results.

In [58]:
```
# Compute average charging power (kW) per session
df['Pavg_kW'] = df['Total_kWh'] / df['ChargingDurationHrs']

# Remove any infinite or NaN average power values (from very short or odd sessio
df = df.replace([np.inf, -np.inf], np.nan).dropna(subset=['Pavg_kW'])

# Set start time as the index for time series analysis
df = df.set_index('StartDateTime').sort_index()

# Resample to hourly intervals (you can change 'H' to '30T' for half-hourly etc.
df_hourly = df.resample('H').agg({
    'Pavg_kW': 'mean',             # average power demand that hour
    'Total_kWh': 'sum',            # total energy delivered that hour
    'ChargingDurationHrs': 'sum',  # total charging time that hour
})

# Preview results
print(df_hourly.head())
```

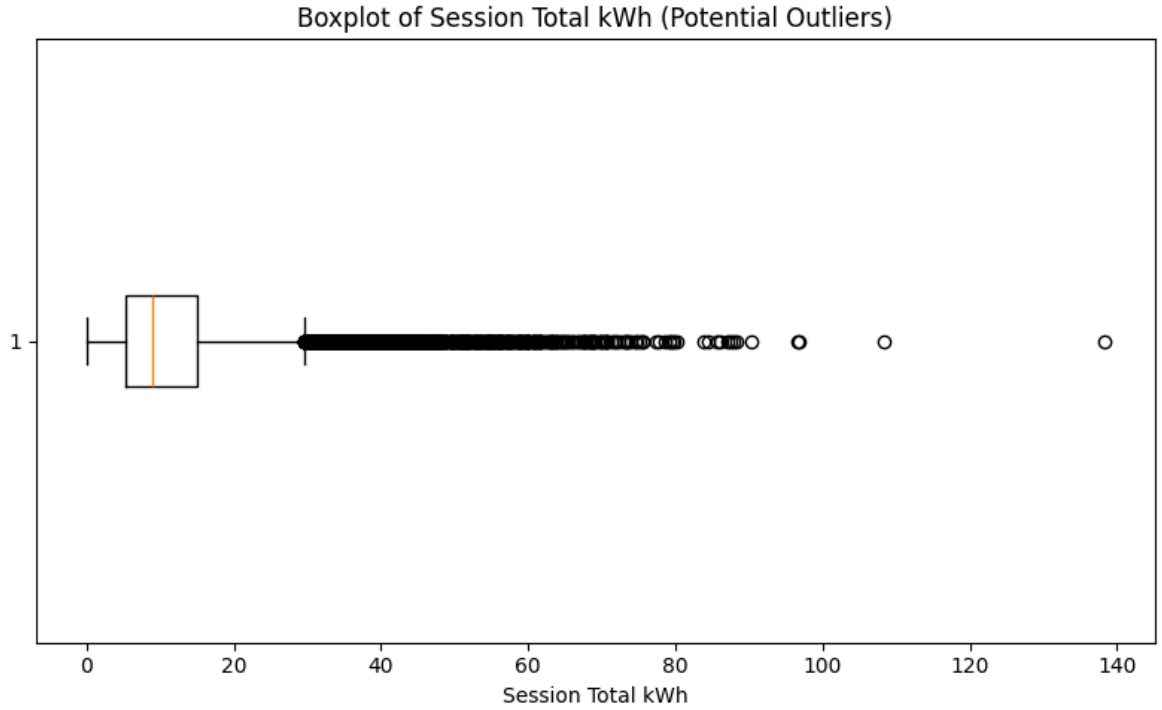
```
                     Pavg_kW  Total_kWh  ChargingDurationHrs
StartDateTime
2016-09-09 07:00:00  25.745455       4.72             0.183333
2016-09-09 08:00:00   7.558800      22.67             6.150000
2016-09-09 09:00:00  14.947826       5.73             0.383333
2016-09-09 10:00:00  15.818182       5.80             0.366667
2016-09-09 11:00:00        NaN       0.00             0.000000
```

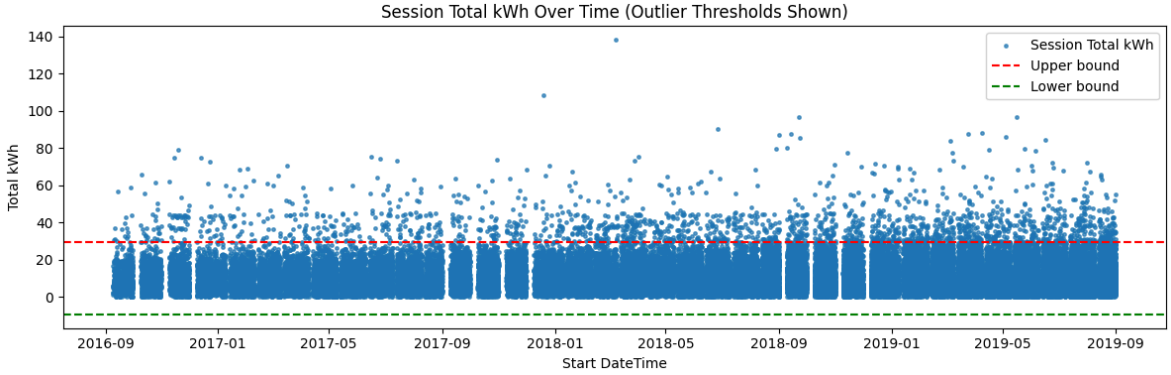
Identifying and Removing Outliers

I am now identifying and removing anomalous data points to improve the quality of my dataset. I first define a valid range for energy consumption per session using the Interquartile Range (IQR) method, a standard statistical technique for outlier detection. To visualize these potential outliers, I generate a boxplot and a scatter plot, which help me confirm the calculated thresholds. After this visual inspection, I filter out any charging sessions that fall outside these established upper and lower bounds. Finally, I print a confirmation of how many outliers were removed and the final size of the cleaned dataset.

In [59]:
```
# Remove non-positive charging before outlier checks

df = df[df['Total_kWh'] > 0]

# Outlier Detection (IQR method) for Total_kWh at session level
Q1 = df['Total_kWh'].quantile(0.25)
```

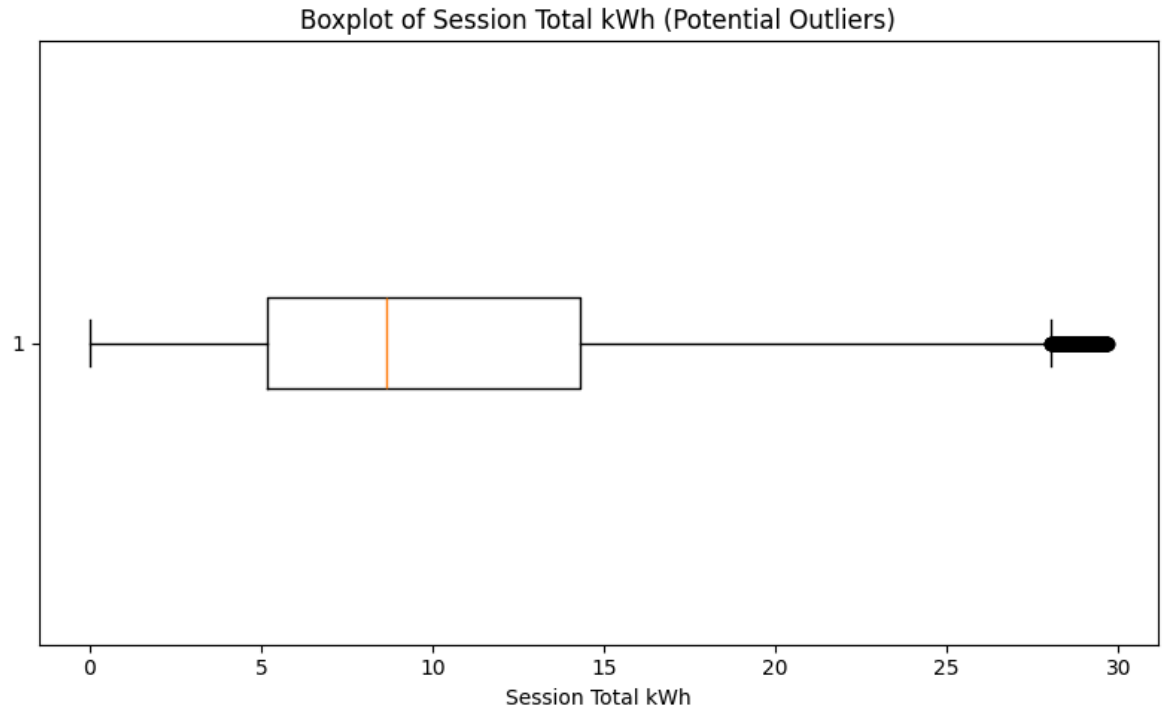
```
Q3 = df['Total_kWh'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Ensure StartDateTime is available and parsed correctly
if 'StartDateTime' not in df.columns:
    df['StartDateTime'] = pd.to_datetime(
        df['Start_Date'].astype(str).str.strip() + ' ' + df['Start_Time'].astype
        dayfirst=True, errors='coerce'
    )
df = df.dropna(subset=['StartDateTime'])

# Visualise session-level outliers before removal
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.boxplot(df['Total_kWh'].dropna(), vert=False)
plt.title('Boxplot of Session Total kWh (Potential Outliers)')
plt.xlabel('Session Total kWh')
plt.tight_layout()
plt.savefig("Boxplot before outlier.png", dpi=1200, bbox_inches='tight')
plt.show()

plt.figure(figsize=(12, 4))
plt.scatter(df['StartDateTime'], df['Total_kWh'], s=6, label='Session Total kWh'
plt.axhline(upper_bound, color='r', linestyle='--', label='Upper bound')
plt.axhline(lower_bound, color='g', linestyle='--', label='Lower bound')
plt.legend()
plt.title('Session Total kWh Over Time (Outlier Thresholds Shown)')
plt.ylabel('Total kWh')
plt.xlabel('Start DateTime')
plt.tight_layout()
plt.savefig("Outliertreshold before outlier.png", dpi=1200, bbox_inches='tight')
plt.show()

# Flag and remove session outliers
n_before = len(df)
df['Outlier'] = ~df['Total_kWh'].between(lower_bound, upper_bound)
n_outliers = df['Outlier'].sum()
print(f"Identified {n_outliers} outlier sessions out of {n_before} ({n_outliers/
df = df[df['Total_kWh'].between(lower_bound, upper_bound)].copy()

# Drop the flag column after filtering
if 'Outlier' in df.columns:
    df = df.drop(columns='Outlier')
df = df.reset_index(drop=True)

print("After cleaning and outlier removal, session-level shape:", df.shape)
```

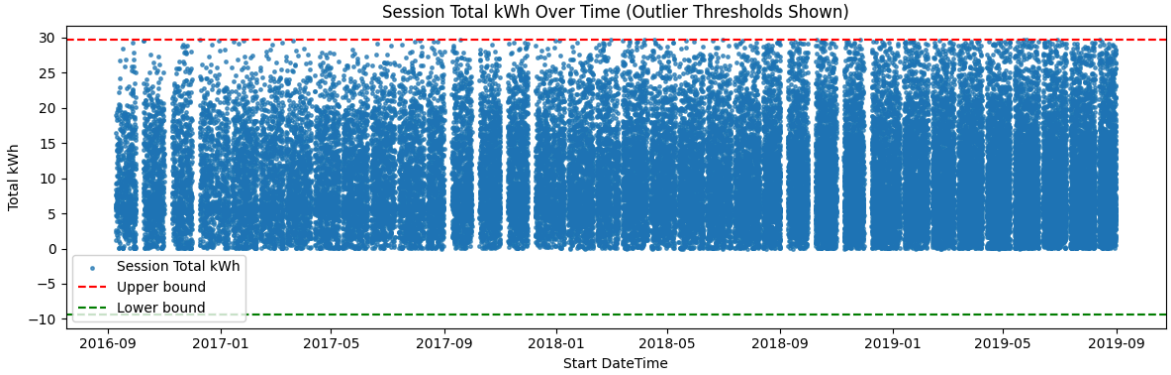





```
Identified 1824 outlier sessions out of 54101 (3.37%)
After cleaning and outlier removal, session-level shape: (52277, 14)
```

In [60]:
```python
# Visualise session-level outliers after removal
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.boxplot(df['Total_kWh'].dropna(), vert=False)
plt.title('Boxplot of Session Total kWh (Potential Outliers)')
plt.xlabel('Session Total kWh')
plt.tight_layout()
plt.savefig("Boxplot after outlier.png", dpi=1200, bbox_inches='tight')
plt.show()

plt.figure(figsize=(12, 4))
plt.scatter(df['StartDateTime'], df['Total_kWh'], s=6, label='Session Total kWh'
plt.axhline(upper_bound, color='r', linestyle='--', label='Upper bound')
plt.axhline(lower_bound, color='g', linestyle='--', label='Lower bound')
plt.legend()
plt.title('Session Total kWh Over Time (Outlier Thresholds Shown)')
plt.ylabel('Total kWh')
plt.xlabel('Start DateTime')
plt.tight_layout()
plt.savefig("Outliertreshold after outlier.png", dpi=1200, bbox_inches='tight')
plt.show()
```





Feature Engineering for Forecasting

To prepare the data for my forecasting model, I am creating several new time-based features from the hourly data. I extract standard calendar variables like the hour, day of the week, month, and year, which will help the model identify cyclical patterns. I also add custom features, such as a "WorkingStatus" indicator to distinguish weekdays from weekends, and a "Season" variable tailored to UK meteorological definitions. Furthermore, I engineer two more complex features: a cumulative hourly average demand and a 24-hour rolling average demand. These are designed to provide the model with a sense of both long-term and recent historical trends. Finally, I remove the initial rows that have incomplete data due to the rolling calculations, ensuring the dataset is clean for the next stage.

In [61]:
```
# Time-Based and Rolling Feature Engineering for EV Load Forecasting

# Hour of day (0-23)
df_hourly['ConnectionHour'] = df_hourly.index.hour

# Day of week (0=Monday, 6=Sunday)
df_hourly['DayofWeek'] = df_hourly.index.dayofweek
```

```
# Week number (ISO standard)
df_hourly['Week'] = df_hourly.index.isocalendar().week.astype(int)

# Month (1-12)
df_hourly['Month'] = df_hourly.index.month

# Year
df_hourly['Year'] = df_hourly.index.year

# Season (UK Meteorological seasons)
def uk_season(month):
    if month in [12, 1, 2]: return 'Winter'
    if month in [3, 4, 5]: return 'Spring'
    if month in [6, 7, 8]: return 'Summer'
    return 'Autumn'
df_hourly['Season'] = df_hourly['Month'].apply(uk_season)

# Working day indicator (1 = Mon-Fri, 0 = Sat/Sun)
df_hourly['WorkingStatus'] = df_hourly['DayofWeek'].apply(lambda x: 1 if x < 5 e

# HourlyAverageDemand (cumulative mean for each hour of day, excluding current h
df_hourly['HourlyAverageDemand'] = (
    df_hourly.groupby('ConnectionHour')['Total_kWh']
    .transform(lambda x: x.expanding().mean().shift(1))
)

# Previous24HrAverageDemand (rolling mean of previous 24 hours, shifted by 1 to
df_hourly['Previous24HrAverageDemand'] = (
    df_hourly['Total_kWh'].shift(1).rolling(window=24, min_periods=1).mean()
)

# Remove rows with NaNs introduced by rolling/expanding calculations
df_hourly = df_hourly.dropna(subset=['HourlyAverageDemand', 'Previous24HrAverage

# Preview the features
print(df_hourly[['ConnectionHour', 'DayofWeek', 'Week', 'Month', 'Year', 'Season
                 'WorkingStatus', 'HourlyAverageDemand', 'Previous24HrAverageDem
```

```
                     ConnectionHour  DayofWeek  Week  Month  Year  Season  Workin
gStatus  HourlyAverageDemand  \
StartDateTime
2016-09-10 07:00:00              7          5    36      9  2016  Autumn
0                4.720
2016-09-10 08:00:00              8          5    36      9  2016  Autumn
0               22.670
2016-09-10 09:00:00              9          5    36      9  2016  Autumn
0                5.730
2016-09-10 10:00:00             10          5    36      9  2016  Autumn
0                5.800
2016-09-10 11:00:00             11          5    36      9  2016  Autumn
0                0.000
2016-09-10 12:00:00             12          5    36      9  2016  Autumn
0                5.810
2016-09-10 13:00:00             13          5    36      9  2016  Autumn
0               30.390
2016-09-10 14:00:00             14          5    36      9  2016  Autumn
0                5.950
2016-09-10 15:00:00             15          5    36      9  2016  Autumn
0               15.220
2016-09-10 16:00:00             16          5    36      9  2016  Autumn
0               21.052

                     Previous24HrAverageDemand
StartDateTime
2016-09-10 07:00:00                   6.291750
2016-09-10 08:00:00                   6.902167
2016-09-10 09:00:00                   6.618417
2016-09-10 10:00:00                   6.379667
2016-09-10 11:00:00                   7.132583
2016-09-10 12:00:00                   7.132583
2016-09-10 13:00:00                   8.550083
2016-09-10 14:00:00                  10.098750
2016-09-10 15:00:00                  10.748375
2016-09-10 16:00:00                  11.969208
```

Data Preparation

I am applying one-hot encoding to the "Season" column. This technique converts the categorical season labels (like 'Winter', 'Spring') into a numerical format that machine learning models can understand, creating a separate binary (0 or 1) column for each season. This is the final preparation step before feeding the data into the forecasting model. I then display the head of the dataframe to confirm the new season columns have been added correctly.

In [62]:
```
# One-hot encode the 'Season' column (creates new columns for each unique season
df_hourly = pd.get_dummies(df_hourly, columns=['Season'], drop_first=True)

df_hourly.head()
```

Out[62]:

StartDateTime	Pavg_kW	Total_kWh	ChargingDurationHrs	ConnectionHour	DayofWee
2016-09-10 07:00:00	19.052459	19.37	1.016667	7	
2016-09-10 08:00:00	15.860000	15.86	1.000000	8	
2016-09-10 09:00:00	NaN	0.00	0.000000	9	
2016-09-10 10:00:00	16.269518	23.87	9.566667	10	
2016-09-10 11:00:00	NaN	0.00	0.000000	11	

In [63]:
```
season_cols = [col for col in df_hourly.columns if col.startswith('Season_')]
df_hourly[season_cols] = df_hourly[season_cols].astype(int)

df_hourly.head()
```

Out[63]:

StartDateTime	Pavg_kW	Total_kWh	ChargingDurationHrs	ConnectionHour	DayofWee
2016-09-10 07:00:00	19.052459	19.37	1.016667	7	
2016-09-10 08:00:00	15.860000	15.86	1.000000	8	
2016-09-10 09:00:00	NaN	0.00	0.000000	9	
2016-09-10 10:00:00	16.269518	23.87	9.566667	10	
2016-09-10 11:00:00	NaN	0.00	0.000000	11	

Exploratory Data Analysis (EDA)

Final Data Verification

Before proceeding to the modeling stage, I am performing a final check on the prepared hourly data. I print the shape to confirm the number of rows and columns, use .describe() to review the statistical summary of my features, and double-check for any remaining missing values to ensure the dataset is complete and clean.

In [64]:
```
# Quick Data Check

print("Hourly DataFrame shape:", df_hourly.shape)
print(df_hourly.describe())
print("\nMissing values per column:\n", df_hourly.isnull().sum())
```

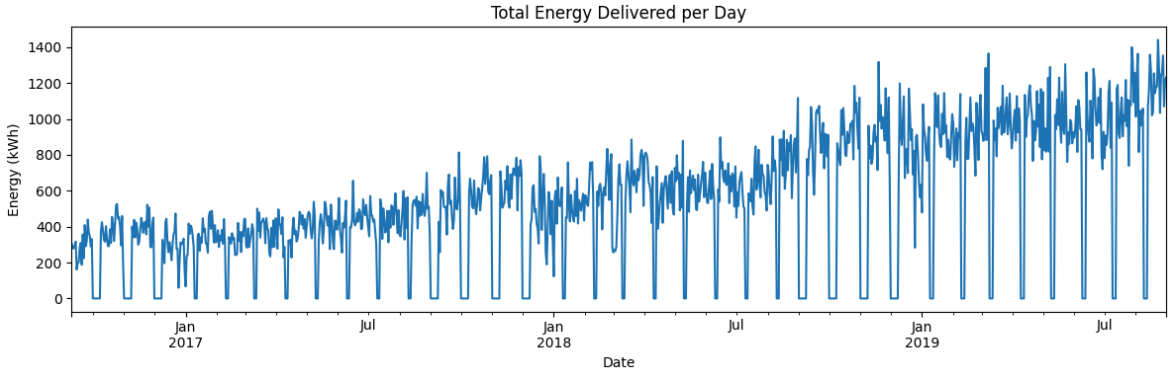
```
Hourly DataFrame shape: (26057, 14)
            Pavg_kW     Total_kWh  ChargingDurationHrs  ConnectionHour     DayofW
eek          Week         Month  \
count  14156.000000  26057.000000         26057.000000    26057.000000  26057.000
000  26057.000000  26057.000000
mean      19.438747     23.012600             3.139173       11.502283      3.001
305     26.445869      6.504855
std       10.267613     32.388725             9.122536        6.921769      2.000
038     15.035114      3.455003
min        0.035077      0.000000             0.000000        0.000000      0.000
000      1.000000      1.000000
25%       13.218309      0.000000             0.000000        6.000000      1.000
000     13.000000      4.000000
50%       18.709780      6.910000             0.366667       12.000000      3.000
000     26.000000      6.000000
75%       24.796337     37.420000             3.416667       18.000000      5.000
000     40.000000     10.000000
max      394.599078    244.226000           481.450000       23.000000      6.000
000     52.000000     12.000000

               Year  WorkingStatus  HourlyAverageDemand  Previous24HrAverageDeman
d  Season_Spring  Season_Summer  \
count  26057.000000   26057.000000         26057.000000               26057.00000
0   26057.000000   26057.000000
mean    2017.680009       0.713820            15.892911                  22.98556
9       0.254212       0.254212
std        0.934284       0.451983            13.332464                  14.22358
3       0.435426       0.435426
min     2016.000000       0.000000             0.000000                   0.00000
0       0.000000       0.000000
25%     2017.000000       0.000000             2.163615                  13.77733
3       0.000000       0.000000
50%     2018.000000       1.000000            15.019717                  22.01508
3       0.000000       0.000000
75%     2018.000000       1.000000            26.938359                  33.77583
3       1.000000       1.000000
max     2019.000000       1.000000            48.974000                  67.37508
3       1.000000       1.000000

       Season_Winter
count   26057.000000
mean        0.248686
std         0.432259
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

Missing values per column:
 Pavg_kW                     11901
Total_kWh                       0
ChargingDurationHrs             0
ConnectionHour                  0
DayofWeek                       0
Week                            0
Month                           0
Year                            0
WorkingStatus                   0
HourlyAverageDemand             0
```

```
Previous24HrAverageDemand        0
Season_Spring                    0
Season_Summer                    0
Season_Winter                    0
dtype: int64
```

Visualising Daily Energy Consumption

I am generating a plot to visualize the total energy delivered across the network each day. By resampling the hourly data to a daily frequency and summing the Total_kWh, I can create a high-level overview of the demand over time. This helps in identifying long-term trends, seasonal patterns, and any significant anomalies in the historical data.

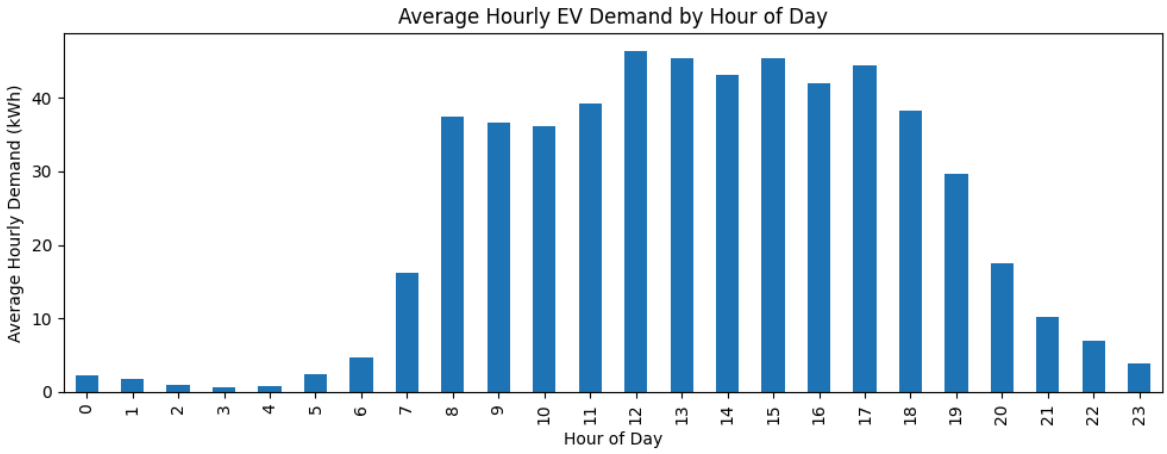
In [65]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,4))
df_hourly['Total_kWh'].resample('D').sum().plot()
plt.title('Total Energy Delivered per Day')
plt.xlabel('Date')
plt.ylabel('Energy (kWh)')
plt.tight_layout()
plt.show()
```



Visualising Average Hourly Demand

I am now creating a bar chart to visualize the average energy demand for each hour of the day. I group the data by the "ConnectionHour" and then calculate the mean Total_kWh for each hour. This visualisation is crucial for understanding the daily demand cycle, clearly showing at which times of day the charging network is most and least busy. I then save the chart as a high-resolution image for use in my dissertation.

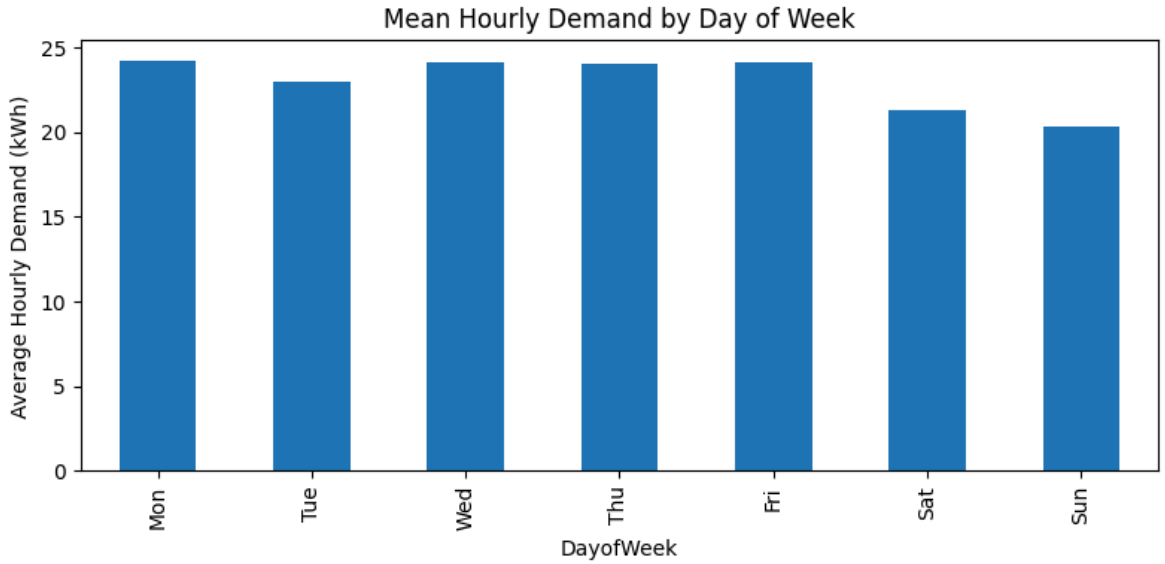
In [66]:
```
plt.figure(figsize=(10,4))
df_hourly.groupby('ConnectionHour')['Total_kWh'].mean().plot(kind='bar')
plt.title('Average Hourly EV Demand by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Average Hourly Demand (kWh)')
plt.tight_layout()
plt.savefig("Average hourly demand.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Visualising Demand by Day of the Week

I am creating a bar chart to show how the average hourly energy demand varies across different days of the week. To make the chart more readable, I first map the numerical day-of-the-week codes to their corresponding names (e.g., 0 to 'Mon'). Then, I group the data by these day names and calculate the mean Total_kWh. This visualisation helps me identify weekly patterns, such as potential differences in charging behavior between weekdays and weekends.

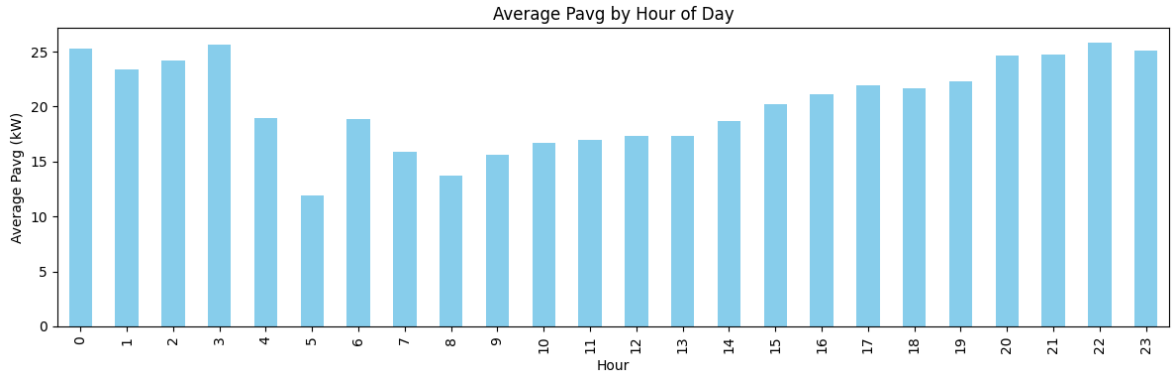
In [67]:
```
dow_map = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}
mean_by_dow = df_hourly.groupby('DayofWeek')['Total_kWh'].mean().rename(index=do

plt.figure(figsize=(8,4))
mean_by_dow.plot(kind='bar')
plt.title('Mean Hourly Demand by Day of Week')
plt.ylabel('Average Hourly Demand (kWh)')
plt.tight_layout()
plt.show()
```



Visualising Average Power by Hour

I am creating a bar chart to examine the average charging power (in kW) for each hour of the day. By grouping the data by the hour and calculating the mean of the average power, this plot helps me identify the times when charging sessions are typically most power-intensive. This provides a different perspective from total energy consumption, focusing instead on the rate of energy transfer.

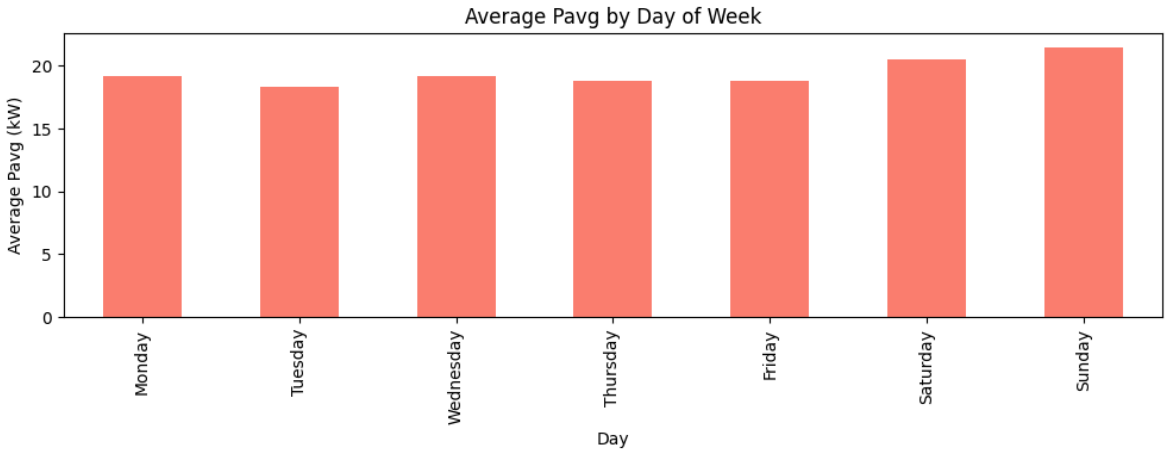
In [68]:
```
# Average Pavg by Hour of Day
plt.figure(figsize=(12,4))
df_hourly.groupby('ConnectionHour')['Pavg_kW'].mean().plot(kind='bar', color='sk
plt.title('Average Pavg by Hour of Day')
plt.xlabel('Hour')
plt.ylabel('Average Pavg (kW)')
plt.tight_layout()
plt.show()
```



Visualising Average Power by Day of the Week

I am generating a bar chart to see how the average charging power changes depending on the day of the week. I first map the numerical day codes to their full names to make the chart labels clear. Then, I group the data by these day names and calculate the mean average power for each. The resulting bar chart allows me to easily compare the typical charging intensity across the week, identifying any weekly usage patterns.

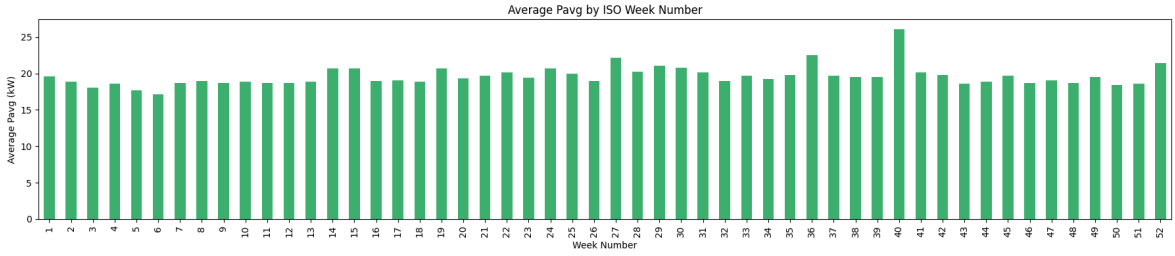
In [70]:
```
# Average Pavg by Day of Week
plt.figure(figsize=(10,4))
dow_map = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday'
avg_by_dow = df_hourly.groupby('DayofWeek')['Pavg_kW'].mean().rename(index=dow_m
avg_by_dow.plot(kind='bar', color='salmon')
plt.title('Average Pavg by Day of Week')
plt.xlabel('Day')
plt.ylabel('Average Pavg (kW)')
plt.tight_layout()
plt.show()
```



Visualising Seasonal Power Trends

I am creating a bar chart to analyze the average charging power across all the weeks of the year. By grouping my data by the ISO week number and plotting the mean of the average power, this visualisation allows me to identify any long-term seasonal trends or variations in charging intensity over the course of a year.

In [71]:
```
# Average Pavg by ISO Week Number
plt.figure(figsize=(18,4))
df_hourly.groupby('Week')['Pavg_kW'].mean().plot(kind='bar', color='mediumseagre
plt.title('Average Pavg by ISO Week Number')
plt.xlabel('Week Number')
plt.ylabel('Average Pavg (kW)')
plt.tight_layout()
plt.show()
```

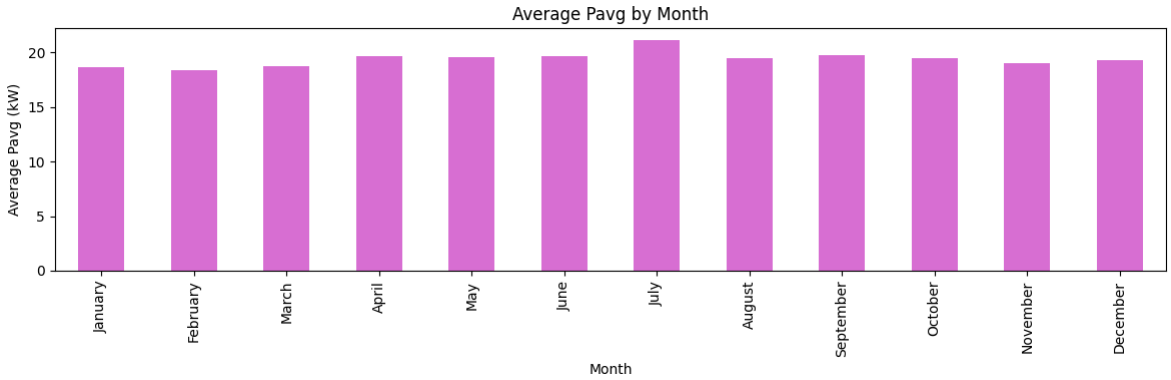


Visualising Monthly Power Trends

I am creating a bar chart to analyze how the average charging power varies throughout the year. To make the visualization clear, I first map the numerical months to their full names. Then, by grouping the data by month and plotting the mean of the average power, I can effectively identify any seasonal patterns or trends in charging intensity.

In [72]:
```
# Average Pavg by Month
plt.figure(figsize=(12,4))
month_map = {1:'January', 2:'February', 3:'March', 4:'April', 5:'May', 6:'June',
avg_by_month = df_hourly.groupby('Month')['Pavg_kW'].mean().rename(index=month_m
avg_by_month.plot(kind='bar', color='orchid')
plt.title('Average Pavg by Month')
plt.xlabel('Month')
plt.ylabel('Average Pavg (kW)')
```

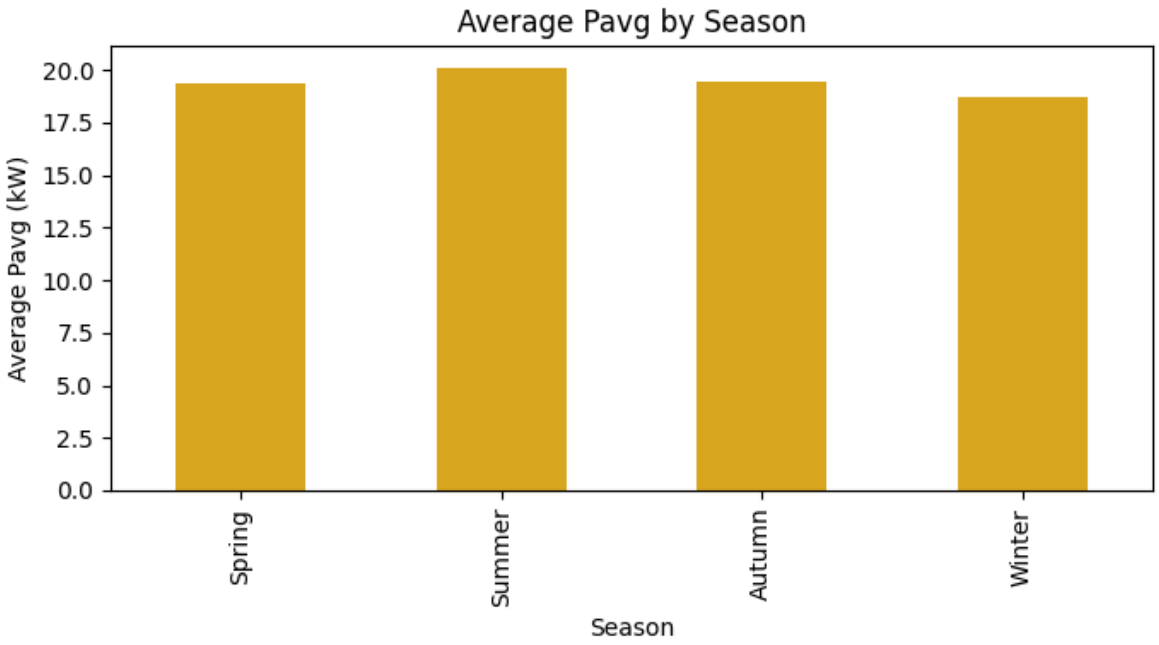
```
plt.tight_layout()
plt.show()
```



Visualising Average Power by Season

To create this visualization, I first need to reverse the one-hot encoding I performed earlier. I create a new "Season_Label" column by interpreting the binary season columns, which allows me to group the data by season again. Once that's done, I plot the mean average power for each season, ordering them logically. This bar chart provides a clear, direct comparison of charging intensity across Spring, Summer, Autumn, and Winter.

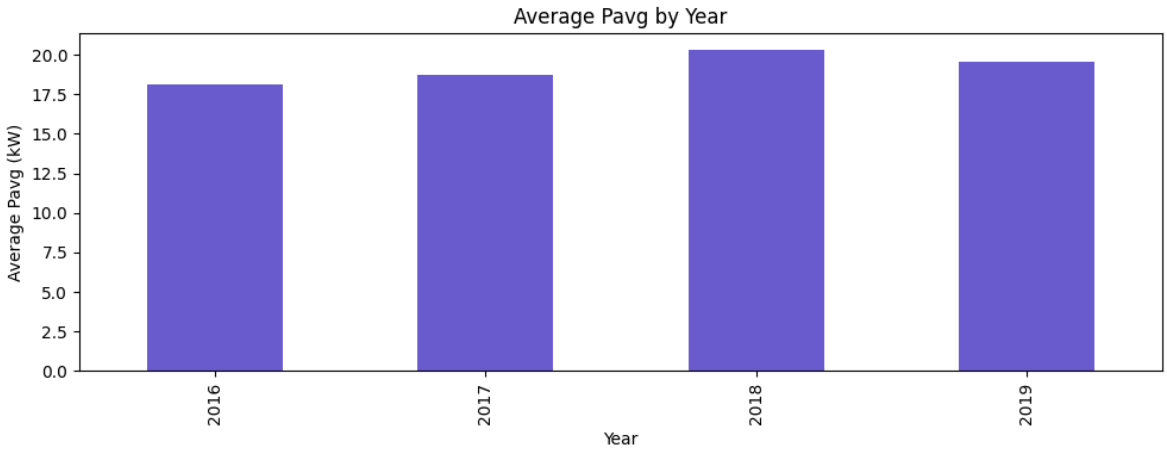
In [73]:
```
# Create a new 'Season_Label' column based on the dummies (assuming you used dro
def infer_season(row):
    if row['Season_Spring'] == 1:
        return 'Spring'
    elif row['Season_Summer'] == 1:
        return 'Summer'
    elif row['Season_Winter'] == 1:
        return 'Winter'
    else:
        return 'Autumn'  # all zeros means Autumn

df_hourly['Season_Label'] = df_hourly.apply(infer_season, axis=1)

plt.figure(figsize=(7,4))
df_hourly.groupby('Season_Label')['Pavg_kW'].mean().loc[['Spring','Summer','Autu
plt.title('Average Pavg by Season')
plt.xlabel('Season')
plt.ylabel('Average Pavg (kW)')
plt.tight_layout()
plt.show()
```

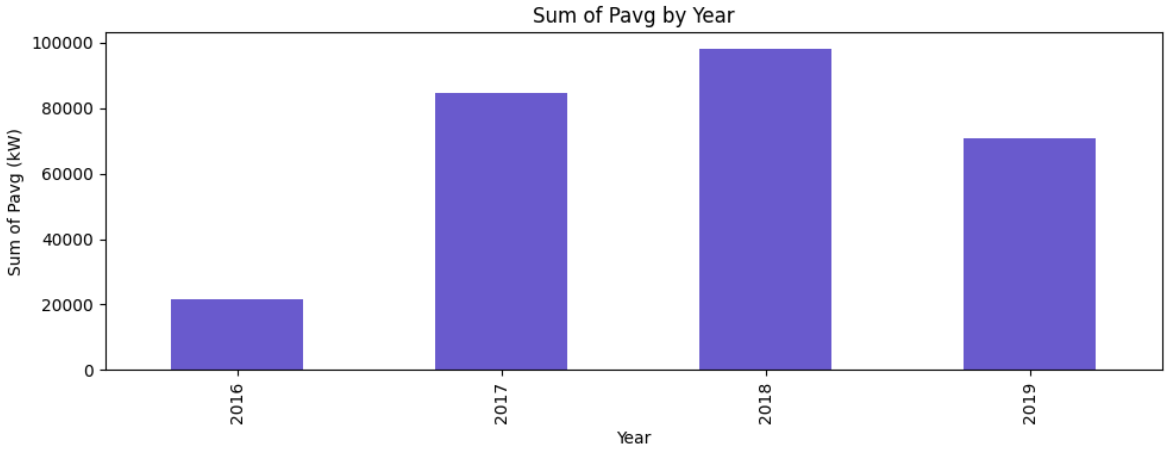


Visualising Year-Over-Year Trends

I am generating two bar charts to analyze trends across the different years in my dataset. The first chart shows the average charging power for each year, which helps me understand if the typical intensity of charging sessions has been changing over time. The second chart visualises the sum of the average power values for each year, providing a clear indication of the overall growth in network usage from one year to the next.

In [74]:
```
# Average Pavg by Year (if multiple years in dataset)
if 'Year' in df_hourly.columns:
    plt.figure(figsize=(10,4))
    df_hourly.groupby('Year')['Pavg_kW'].mean().plot(kind='bar', color='slateblu
    plt.title('Average Pavg by Year')
    plt.xlabel('Year')
    plt.ylabel('Average Pavg (kW)')
    plt.tight_layout()
    plt.show()

    # Sum of Pavg by Year
    plt.figure(figsize=(10,4))
    df_hourly.groupby('Year')['Pavg_kW'].sum().plot(kind='bar', color='slateblue
    plt.title('Sum of Pavg by Year')
    plt.xlabel('Year')
    plt.ylabel('Sum of Pavg (kW)')
    plt.tight_layout()
    plt.show()
```





In [75]:
```
!pip install statsmodels
```

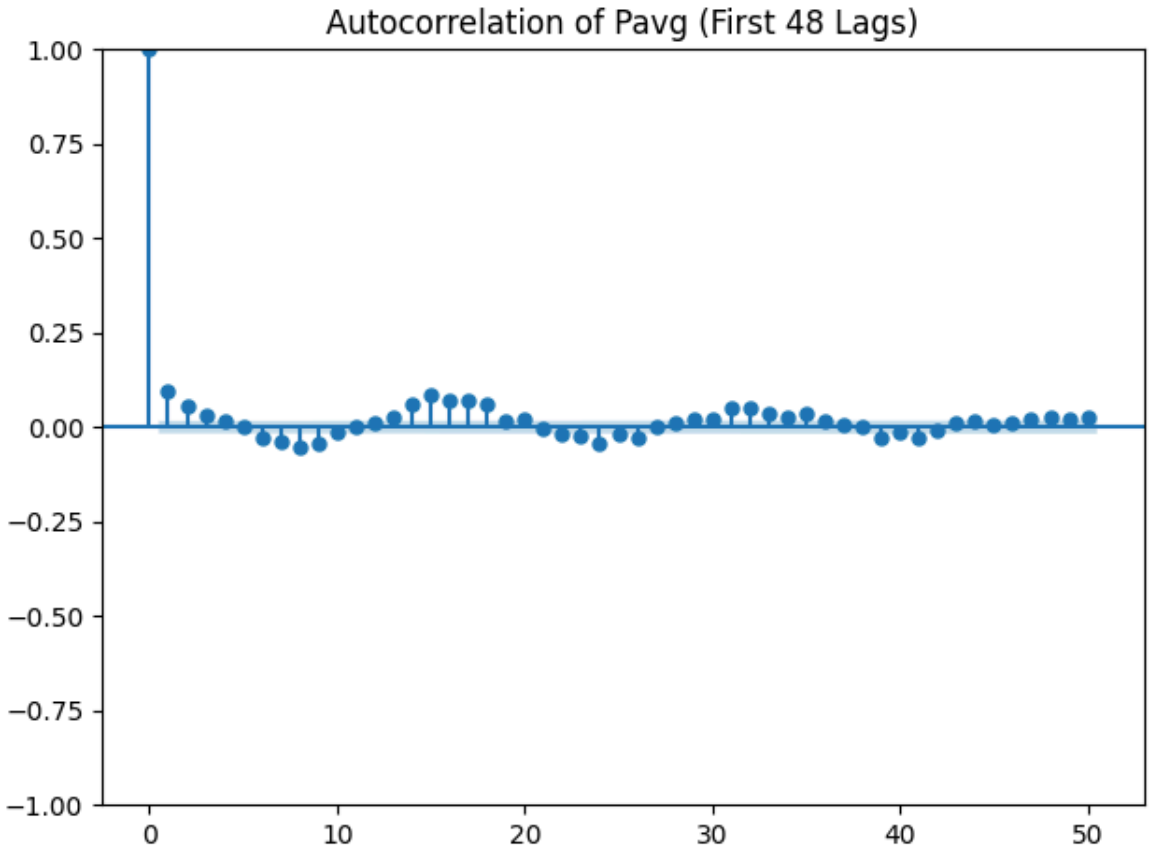
```
Collecting statsmodels
  Downloading statsmodels-0.14.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_
x86_64.manylinux_2_28_x86_64.whl.metadata (9.5 kB)
Requirement already satisfied: numpy<3,>=1.22.3 in /usr/local/lib/python3.11/dist
-packages (from statsmodels) (2.0.2)
Requirement already satisfied: scipy!=1.9.2,>=1.8 in /usr/local/lib/python3.11/di
st-packages (from statsmodels) (1.16.1)
Requirement already satisfied: pandas!=2.1.0,>=1.4 in /usr/local/lib/python3.11/d
ist-packages (from statsmodels) (2.2.2)
Collecting patsy>=0.5.6 (from statsmodels)
  Downloading patsy-1.0.1-py2.py3-none-any.whl.metadata (3.3 kB)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.11/dist-
packages (from statsmodels) (25.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
1/dist-packages (from pandas!=2.1.0,>=1.4->statsmodels) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas!=2.1.0,>=1.4->statsmodels) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil>=2.8.2->pandas!=2.1.0,>=1.4->statsmodels) (1.17.0)
Downloading statsmodels-0.14.5-cp311-cp311-manylinux2014_x86_64.manylinux_2_17_x8
6_64.manylinux_2_28_x86_64.whl (10.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.5/10.5 MB 91.3 MB/s eta 0:00:00
Downloading patsy-1.0.1-py2.py3-none-any.whl (232 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 232.9/232.9 kB 22.0 MB/s eta 0:00:00
Installing collected packages: patsy, statsmodels
Successfully installed patsy-1.0.1 statsmodels-0.14.5
```

Analysing Temporal Dependencies

I am generating an autocorrelation plot for the average power (Pavg_kW) to understand its temporal structure. This plot measures the correlation of the time series with itself at different time lags. By examining these correlations for the past 50 hours, I can identify significant repeating patterns, such as a strong 24-hour cycle, which is a crucial step in diagnosing the data's characteristics before building a forecasting model.

In [76]:
```
# Autocorrelation Plot of Pavg
from statsmodels.graphics.tsaplots import plot_acf

# Plot autocorrelation for the first 48 lags (2 days for hourly data)
plt.figure(figsize=(7,5))
plot_acf(df_hourly['Pavg_kW'].dropna(), lags=50)
plt.title('Autocorrelation of Pavg (First 48 Lags)')
plt.tight_layout()
plt.show()
```

```
<Figure size 700x500 with 0 Axes>
```



How to Interpret This Plot

1. Spike at Lag 0

The value is always 1 at lag 0—this is just the correlation of the series with itself.

2. Lag 1 to Lag 48

You see the autocorrelation coefficient for lags 1 to 50.

Each point is the correlation between the series and itself shifted by that many hours.

3. What the shape means:

The pattern is a gentle oscillation: small positive and negative autocorrelations that repeat with a moderate period (about 24 hours, as you can see from a slight cycle).

No extremely strong daily (24h) or weekly (168h) repeating structure, but there is weak seasonality—visible as the small “bumps” near every 24 lags.

The autocorrelation values are mostly near zero except for the first few lags, which means:

There is some short-term correlation: The value at hour t is weakly related to recent hours (this is typical in real energy or demand series, where consumption or demand is “sticky”).

But the series is not dominated by strong regular cycles (e.g. if this were household electricity, you'd see a huge peak at 24, 48, etc.; your workplace/public EV demand is more diffused).

4. Stationarity

The lack of strong upward or downward trend in the autocorrelation coefficients (after lag 0) suggests your series is fairly stationary in the mean.

What Does This Mean for Forecasting? Some benefit to including lagged demand features: e.g., using demand at t-1, t-2 as predictors.

Possible value in capturing daily effects: Rolling features or periodic encodings (hour of day, day of week) could help.

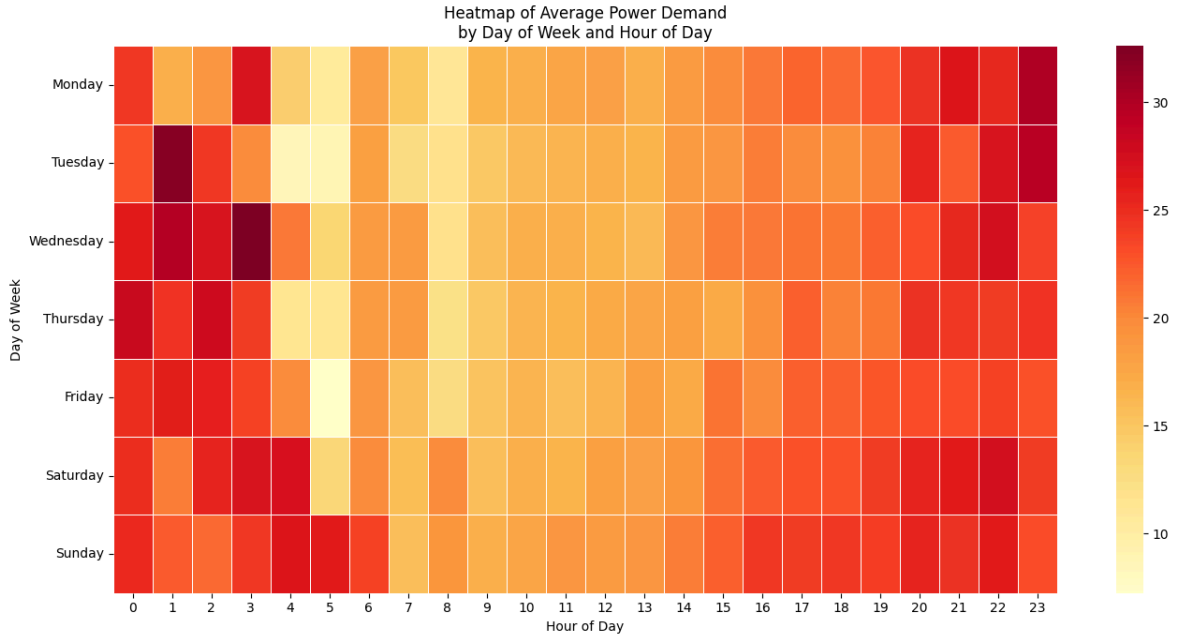
No dominant seasonality: Your EV load is not simply repeating the same pattern every day/hour, likely due to workplace and public station context.

Visualising Weekly Demand Patterns

I am creating a pivot table to structure the average power demand with the days of the week as rows and hours of the day as columns. I then use this table to generate a heatmap, which provides a clear, colour-coded visualisation of when charging demand is at its highest and lowest. This allows me to easily spot patterns, such as peak times during weekdays versus different usage on weekends, all within a single, intuitive chart. Finally, I customise the labels to ensure the chart is easy to interpret.

In [77]:
```
# Heatmap of Average Power Demand by Day of Week and Hour of Day
import seaborn as sns

pivot = df_hourly.pivot_table(index='DayofWeek', columns='ConnectionHour', value
plt.figure(figsize=(14,7))
sns.heatmap(pivot, cmap='YlOrRd', linewidths=0.5)
plt.title('Heatmap of Average Power Demand\nby Day of Week and Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Day of Week')
plt.yticks(ticks=np.arange(0.5, 7.5), labels=['Monday','Tuesday','Wednesday','Th
```

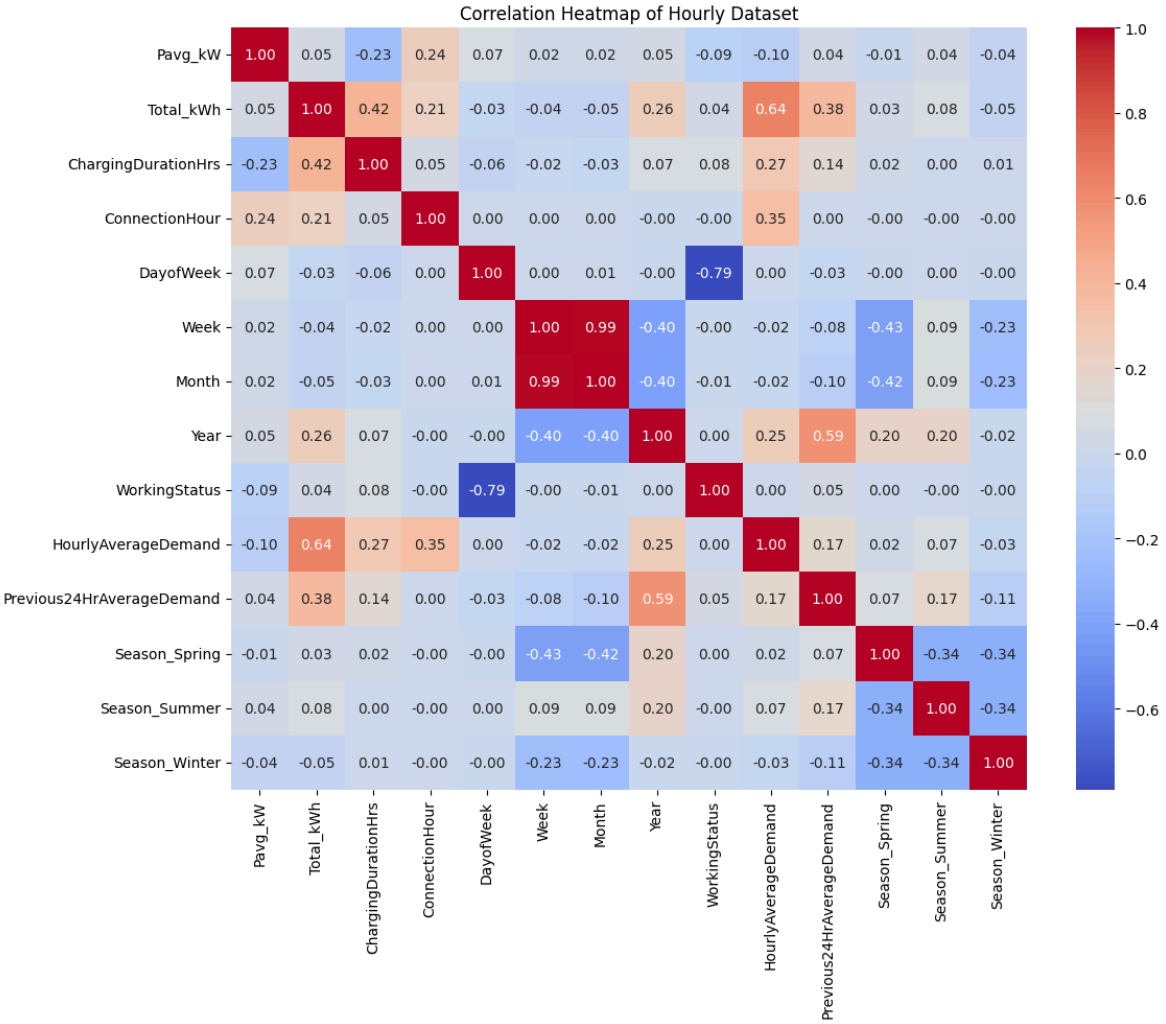
```
plt.tight_layout()
plt.show()
```



Analysing Feature Relationships

I am generating a correlation heatmap to understand the relationships between all the numerical features in my dataset. First, I select only the columns containing numerical data and calculate their correlation matrix. I then use this matrix to create a heatmap, which provides a colour-coded grid where each cell shows the correlation between two features. I have also annotated the map with the specific correlation values to make it easier to interpret. This visualisation is a powerful tool for identifying which features are strongly related to each other, which is an important consideration for my forecasting model. Finally, I save a high-resolution version of this chart for my dissertation.

In [78]:
```
# Correlation Heatmap (of all numeric features)
plt.figure(figsize=(12,10))
num_cols = df_hourly.select_dtypes(include=np.number).columns
corr = df_hourly[num_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap of Hourly Dataset')
plt.tight_layout()
plt.savefig("Heatmap.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Features to Drop (with Reasoning)

1. Target column 'Total_kWh' – This is your target, so must not be included as a feature.

2. Highly correlated features (choose only one for each highly-correlated group) 'Month' and 'Week' – Both are highly correlated with each other and with 'Year'. If your aim is short-term load forecasting, you typically keep only one or drop all if you have more granular time/season features (like 'Season').

'Year' – Only needed if you expect a strong trend; otherwise, can drop to prevent collinearity.

3. Duplicates or categorical features already captured elsewhere 'DayOfWeek' – If you keep 'WorkingStatus' (which is a binarised/aggregated version), drop the original.

Any "Code" columns created for encoding if you use one-hot or ordinal encoding elsewhere.

'ChargingDurationHrs' – Unless justified by your research question, as it's a function of the total kWh and session times. If you keep only demand forecasting, you can drop this.

4. Pavg_kW 'Pavg_kW' – Average power per hour; since you are forecasting energy, not power, this may be dropped unless used as a useful lagged feature.

Encoding Day of the Week

I am now converting the numerical DayofWeek column into a format that is more suitable for machine learning models. First, I map the numbers (0-6) to their corresponding day names for clarity. Then, I apply one-hot encoding to these names, which creates new binary columns for each day of the week. This allows my model to treat each day as a distinct category without assuming any numerical relationship between them. Finally, I remove the original DayofWeek column as it is now redundant.

In [79]:
```
# First, let's map the day of the week numbers to names for clarity in the new c
dow_map = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}
df_hourly['DayofWeek_Name'] = df_hourly['DayofWeek'].map(dow_map)

# Now, one-hot encode the new 'DayofWeek_Name' column.
# drop_first=True is good practice for linear models to avoid multicollinearity.
df_hourly = pd.get_dummies(df_hourly, columns=['DayofWeek_Name'], prefix='Day',

# You can now drop the original numeric 'DayofWeek' column as it's redundant.
if 'DayofWeek' in df_hourly.columns:
    df_hourly = df_hourly.drop(columns=['DayofWeek'])

print("DataFrame columns after one-hot encoding Day of Week:")
print(df_hourly.columns)
```

```
DataFrame columns after one-hot encoding Day of Week:
Index(['Pavg_kW', 'Total_kWh', 'ChargingDurationHrs', 'ConnectionHour', 'Week',
'Month', 'Year', 'WorkingStatus',
       'HourlyAverageDemand', 'Previous24HrAverageDemand', 'Season_Spring', 'Seas
on_Summer', 'Season_Winter',
       'Season_Label', 'Day_Mon', 'Day_Sat', 'Day_Sun', 'Day_Thu', 'Day_Tue', 'Da
y_Wed'],
      dtype='object')
```

Defining Features and Target

I am now finalising the dataset for the modelling stage. I am formally separating my data into two distinct components: the feature matrix (X), which contains all the input variables the model will learn from, and the target vector (y), which is the Total_kWh that I want to predict. To create the feature matrix, I explicitly drop the target variable and any other columns that are not required as inputs, such as redundant date parts or the original categorical labels. Finally, I print the list of remaining columns in my feature matrix to confirm that the separation has been performed correctly.

In [80]:
```
# Define the list of columns you want to drop to create your features.
features_to_drop = [
    'Total_kWh',              # The target variable
    'Pavg_kw',
    'Month',
    'Week',
    'Year',
    'ChargingDurationHrs',
]
```

```
# Add the categorical 'Season_Label' if it exists
if 'Season_Label' in df_hourly.columns:
    features_to_drop.append('Season_Label')


# Define your target vector 'y' from the original DataFrame
y = df_hourly['Total_kWh']

# Define your feature matrix 'X' by dropping the columns from the original DataF
X = df_hourly.drop(columns=[col for col in features_to_drop if col in df_hourly.

# Verify the final feature set
print("Final feature columns in X:")
print(X.columns.tolist())
```

```
Final feature columns in X:
['Pavg_kW', 'ConnectionHour', 'WorkingStatus', 'HourlyAverageDemand', 'Previous24
HrAverageDemand', 'Season_Spring', 'Season_Summer', 'Season_Winter', 'Day_Mon',
'Day_Sat', 'Day_Sun', 'Day_Thu', 'Day_Tue', 'Day_Wed']
```

In [81]:
```
print("Feature columns in X:")
print(X.columns.tolist())

if 'Season_Label' in X.columns:
    X = X.drop(columns=['Season_Label'])

print(X.tail())
```

```
Feature columns in X:
['Pavg_kW', 'ConnectionHour', 'WorkingStatus', 'HourlyAverageDemand', 'Previous24
HrAverageDemand', 'Season_Spring', 'Season_Summer', 'Season_Winter', 'Day_Mon',
'Day_Sat', 'Day_Sun', 'Day_Thu', 'Day_Tue', 'Day_Wed']
                     Pavg_kW  ConnectionHour  WorkingStatus  HourlyAverageDeman
d  Previous24HrAverageDemand  \
StartDateTime
2019-08-31 19:00:00  15.165680              19              0            29.63005
2                  51.058875
2019-08-31 20:00:00  18.348286              20              0            17.42423
5                  50.309125
2019-08-31 21:00:00  17.100000              21              0            10.23806
6                  52.192708
2019-08-31 22:00:00        NaN              22              0             7.01316
5                  52.003542
2019-08-31 23:00:00  27.385714              23              0             3.81916
1                  51.103625

                     Season_Spring  Season_Summer  Season_Winter  Day_Mon  Day_Sa
t  Day_Sun  Day_Thu  Day_Tue  Day_Wed
StartDateTime
2019-08-31 19:00:00              0              1              0    False     Tru
e    False    False    False    False
2019-08-31 20:00:00              0              1              0    False     Tru
e    False    False    False    False
2019-08-31 21:00:00              0              1              0    False     Tru
e    False    False    False    False
2019-08-31 22:00:00              0              1              0    False     Tru
e    False    False    False    False
2019-08-31 23:00:00              0              1              0    False     Tru
e    False    False    False    False
```

Converting Data Types

I am identifying any columns in my feature set that have a boolean (True/False) data type and converting them into an integer format, where True becomes 1 and False becomes 0. This is a final standardisation step to ensure all my input features are numerical, which is a requirement for the machine learning algorithms I will be using. I then display the last few rows of the data to confirm that the changes have been applied correctly.

In [82]:
```
# Identify all columns with the boolean data type
bool_columns = X.select_dtypes(include='bool').columns

# Convert those columns to integer type (True becomes 1, False becomes 0)
X[bool_columns] = X[bool_columns].astype(int)

# Display the end of the DataFrame to confirm the change
print(X.tail())
```

```
                     Pavg_kW  ConnectionHour  WorkingStatus  HourlyAverageDeman
d  Previous24HrAverageDemand  \
StartDateTime
2019-08-31 19:00:00  15.165680              19              0            29.63005
2                  51.058875
2019-08-31 20:00:00  18.348286              20              0            17.42423
5                  50.309125
2019-08-31 21:00:00  17.100000              21              0            10.23806
6                  52.192708
2019-08-31 22:00:00        NaN              22              0             7.01316
5                  52.003542
2019-08-31 23:00:00  27.385714              23              0             3.81916
1                  51.103625

                    Season_Spring  Season_Summer  Season_Winter  Day_Mon  Day_Sa
t  Day_Sun  Day_Thu  Day_Tue  Day_Wed
StartDateTime
2019-08-31 19:00:00              0              1              0        0
1        0        0        0        0
2019-08-31 20:00:00              0              1              0        0
1        0        0        0        0
2019-08-31 21:00:00              0              1              0        0
1        0        0        0        0
2019-08-31 22:00:00              0              1              0        0
1        0        0        0        0
2019-08-31 23:00:00              0              1              0        0
1        0        0        0        0
```

Splitting the Data for Model Evaluation

To properly evaluate my forecasting model, I am splitting the data chronologically into three distinct sets. I first ensure the data is sorted by time. Then, I allocate the first 70% of the data for training the model, the next 15% as a validation set for tuning the model, and the final 15% as a test set. This final test set remains entirely unseen by the model during training and tuning, allowing for an unbiased assessment of its real-world

performance. Finally, I print the dimensions of each set to confirm the split was successful.

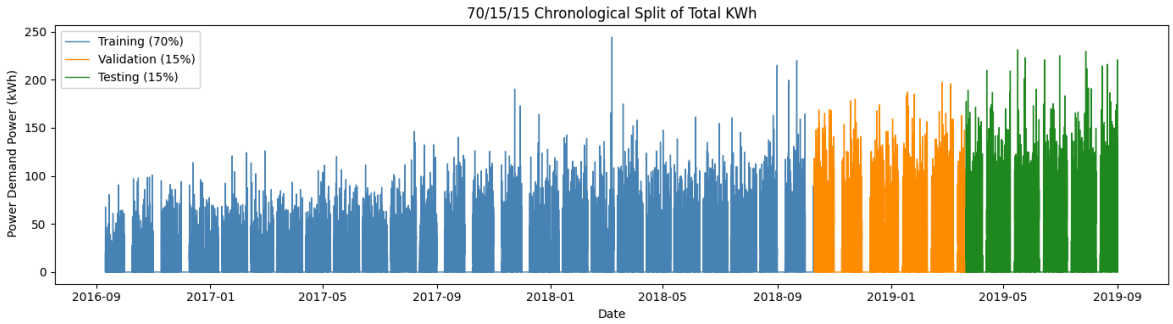
In [83]:
```
# Ensure data is sorted by time
X = X.sort_index()
y = y.sort_index()

n = len(X)
train_size = int(n * 0.7)
val_size = int(n * 0.15)
test_size = n - train_size - val_size

X_train = X.iloc[:train_size]
y_train = y.iloc[:train_size]
X_val = X.iloc[train_size:train_size + val_size]
y_val = y.iloc[train_size:train_size + val_size]
X_test = X.iloc[train_size + val_size:]
y_test = y.iloc[train_size + val_size:]

print("Train:", X_train.shape)
print("Val:", X_val.shape)
print("Test:", X_test.shape)
```

```
Train: (18239, 14)
Val: (3908, 14)
Test: (3910, 14)
```

Visualising the Data Split

I am creating a plot to visualise how my time series data has been divided into the training, validation, and test sets. By plotting each of the three sets in a different colour on the same timeline, I can clearly see the chronological split. This graph serves as a crucial visual confirmation that the data has been partitioned correctly before I proceed with model training. Finally, I customise the plot with a title and labels and save it as a high-resolution image for my dissertation.

In [84]:
```
# Plotting
plt.figure(figsize=(14, 4))
plt.plot(y_train.index, y_train, label='Training (70%)', color='steelblue', line
plt.plot(y_val.index, y_val, label='Validation (15%)', color='darkorange', linew
plt.plot(y_test.index, y_test, label='Testing (15%)', color='forestgreen', linew

# Customisation
plt.title("70/15/15 Chronological Split of Total KWh")
plt.xlabel("Date")
plt.ylabel(" Power Demand Power (kWh)")
plt.legend()
plt.tight_layout()

# Save high-quality figure
plt.savefig("pavg_split_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Handling Missing Data

To ensure my model runs without errors, I am now addressing any gaps in the data that may have resulted from the feature engineering steps. I am using an imputer to fill any missing values with the mean of the respective column. Crucially, I calculate this mean using only the training data to prevent any information from the validation or test sets leaking into the training process. I then apply this same learned imputation to all three data splits—training, validation, and test—to ensure consistent processing. Finally, I run a check to confirm that no missing values remain in any of the datasets.

In [85]:
```
from sklearn.impute import SimpleImputer
import pandas as pd

# Create the imputer
imputer = SimpleImputer(strategy='mean')

# Fit the imputer on the training data and transform X_train
print("Imputing NaNs in X_train...")
# We save the columns/index because transform() returns a NumPy array
X_train_cols = X_train.columns
X_train_idx = X_train.index
X_train = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train_cols, ind

# Transform X_val and X_test using the same fitted imputer
print("Imputing NaNs in X_val and X_test...")
X_val_cols = X_val.columns
X_val_idx = X_val.index
X_val = pd.DataFrame(imputer.transform(X_val), columns=X_val_cols, index=X_val_i

X_test_cols = X_test.columns
X_test_idx = X_test.index
X_test = pd.DataFrame(imputer.transform(X_test), columns=X_test_cols, index=X_te

print("\nImputation complete. Checking for remaining NaNs:")
print(f"NaNs in X_train: {X_train.isnull().sum().sum()}")
print(f"NaNs in X_val: {X_val.isnull().sum().sum()}")
print(f"NaNs in X_test: {X_test.isnull().sum().sum()}")
```

```
Imputing NaNs in X_train...
Imputing NaNs in X_val and X_test...

Imputation complete. Checking for remaining NaNs:
NaNs in X_train: 0
NaNs in X_val: 0
NaNs in X_test: 0
```

Normalising the Data

I am now scaling all my data to ensure that every feature has a consistent range, which is a crucial step for many machine learning algorithms. Using a MinMaxScaler, I am normalising both my feature sets (X) and my target variable (y) to a value between 0 and 1. To prevent any data leakage, I first fit the scaler exclusively on my training data. I then use this same fitted scaler to transform the training, validation, and test sets, ensuring that the same scaling parameters are applied consistently across all my data splits.

In [86]:
```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd

# --- 1. Scale the Feature Sets (X) ---
scaler_X = MinMaxScaler()

# Fit the scaler on the training data ONLY, then transform all three sets
print("Scaling feature sets (X)...")
X_train_scaled = scaler_X.fit_transform(X_train)
X_val_scaled = scaler_X.transform(X_val)
X_test_scaled = scaler_X.transform(X_test) # <-- This line creates X_test_scaled


# --- 2. Scale the Target Sets (y) ---
scaler_y = MinMaxScaler()

# Fit the scaler on the training target ONLY, then transform all three sets
print("Scaling target sets (y)...")
y_train_scaled = scaler_y.fit_transform(y_train.to_numpy().reshape(-1, 1))
y_val_scaled = scaler_y.transform(y_val.to_numpy().reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.to_numpy().reshape(-1, 1)) # <-- This


print("\nAll data splits have been successfully scaled.")
print(f"Shape of X_train_scaled: {X_train_scaled.shape}")
print(f"Shape of X_test_scaled: {X_test_scaled.shape}")
```

```
Scaling feature sets (X)...
Scaling target sets (y)...

All data splits have been successfully scaled.
Shape of X_train_scaled: (18239, 14)
Shape of X_test_scaled: (3910, 14)
```

Training a Baseline Model

As a baseline for comparison, I am now training a standard Linear Regression model. I first train, or 'fit', the model using my prepared training data. Once trained, I use it to make predictions on both the validation and the unseen test datasets. Finally, to judge its performance, I calculate three key metrics for both sets: the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE), and the R-squared value. This gives me a clear, quantitative measure of how accurately this basic model can predict energy demand.

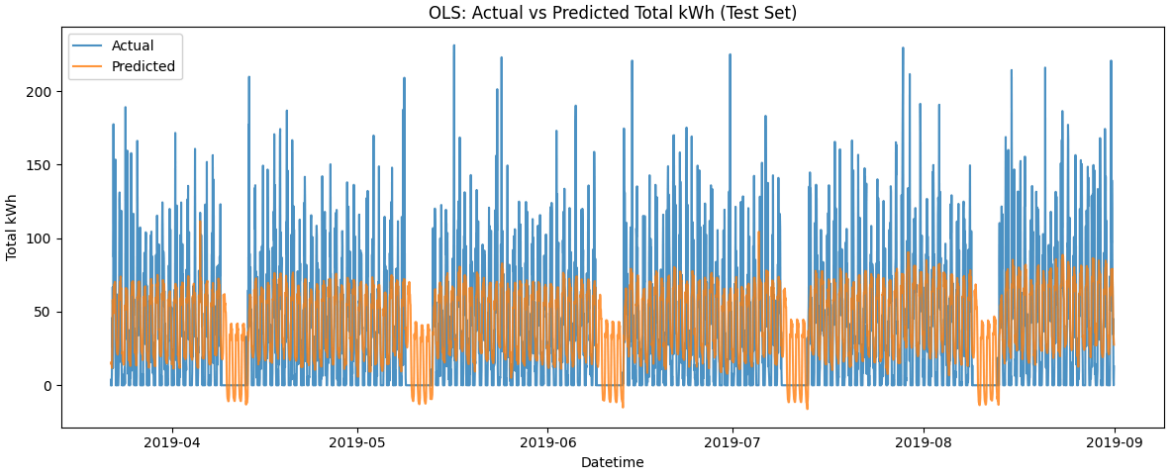
In [87]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1. Instantiate and fit the model
lr = LinearRegression()
lr.fit(X_train, y_train)

# 2. Predict on validation and test sets
y_val_pred = lr.predict(X_val)
y_test_pred = lr.predict(X_test)

# 3. Evaluate performance
def print_metrics(y_true, y_pred, set_name='Set'):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred) ** 0.5
    r2 = r2_score(y_true, y_pred)
    print(f"{set_name} MAE: {mae:.2f}")
    print(f"{set_name} RMSE: {rmse:.2f}")
    print(f"{set_name} R^2: {r2:.3f}")

print_metrics(y_val, y_val_pred, 'Validation')
print_metrics(y_test, y_test_pred, 'Test')
```

```
Validation MAE: 22.12
Validation RMSE: 28.50
Validation R^2: 0.479
Test MAE: 24.05
Test RMSE: 31.29
Test R^2: 0.490
```

Visualising Model Performance

To visually assess the performance of my Linear Regression model, I am plotting its predictions against the actual values from the test set. I create a line graph that shows the true energy demand alongside the demand predicted by the model for the same time period. This provides an intuitive way to see how well the model's forecasts track the real-world data. Finally, I add a title and labels for clarity and save the visualisation as a high-resolution image for my dissertation.

In [88]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test, label='Actual', alpha=0.8)
plt.plot(y_test.index, y_test_pred, label='Predicted', alpha=0.8)
plt.legend()
plt.title('OLS: Actual vs Predicted Total kWh (Test Set)')
plt.xlabel('Datetime')
plt.ylabel('Total kWh')
plt.tight_layout()
plt.savefig("OLS_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Training a Statistical Time-Series Model

I am now implementing a SARIMA model, which is a powerful statistical method specifically designed for time-series data that has seasonal patterns. I have configured the model to recognise a daily seasonality by setting the seasonal period to 24 hours. After fitting the model to the training data, I generate forecasts for both the validation and test periods. Finally, I evaluate its performance using the same set of metrics as the baseline model to provide a direct comparison.

In [89]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

# Try with seasonal order (e.g., daily seasonality, s=24)
model = SARIMAX(y_train, order=(2,0,2), seasonal_order=(1,1,1,24))
sarima_fit = model.fit()
y_val_pred_sarima = sarima_fit.forecast(steps=len(y_val))
y_test_pred_sarima = sarima_fit.forecast(steps=len(y_val)+len(y_test))[-len(y_te
print_metrics(y_val, y_val_pred_sarima, 'SARIMA Validation')
print_metrics(y_test, y_test_pred_sarima, 'SARIMA Test')
```

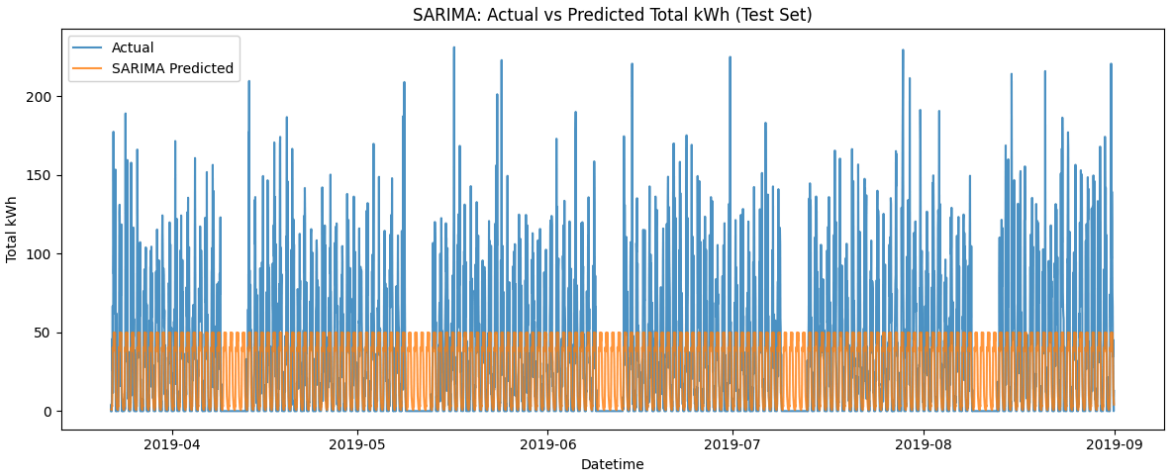
```
SARIMA Validation MAE: 21.18
SARIMA Validation RMSE: 31.96
SARIMA Validation R^2: 0.345
SARIMA Test MAE: 24.01
SARIMA Test RMSE: 36.79
SARIMA Test R^2: 0.295
```

Visualising SARIMA Model Performance

To visually evaluate the SARIMA model, I am plotting its forecasts directly against the actual Total_kWh values from the test set. This line graph provides a clear, side-by-side comparison, allowing me to intuitively judge how well the model's predictions align with the real-world data and capture its patterns. I then add a title and labels for clarity before saving the final visualisation as a high-resolution image for my dissertation.

In [53]:
```
plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test, label='Actual', alpha=0.8)
plt.plot(y_test.index, y_test_pred_sarima, label='SARIMA Predicted', alpha=0.8)
plt.legend()
plt.title('SARIMA: Actual vs Predicted Total kWh (Test Set)')
```

```
plt.xlabel('Datetime')
plt.ylabel('Total kWh')
plt.tight_layout()
plt.savefig("SARIMA_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```



"The SARIMA model predicts a smoothed, periodic pattern that fails to capture the pronounced peaks and high variability in actual EV charging demand. This underfitting is typical for linear time series models when applied to highly non-stationary and event-driven datasets. The model's inability to adapt to rapid changes or spikes highlights the need for advanced machine learning approaches that can leverage engineered features and capture more complex, non-linear relationships."

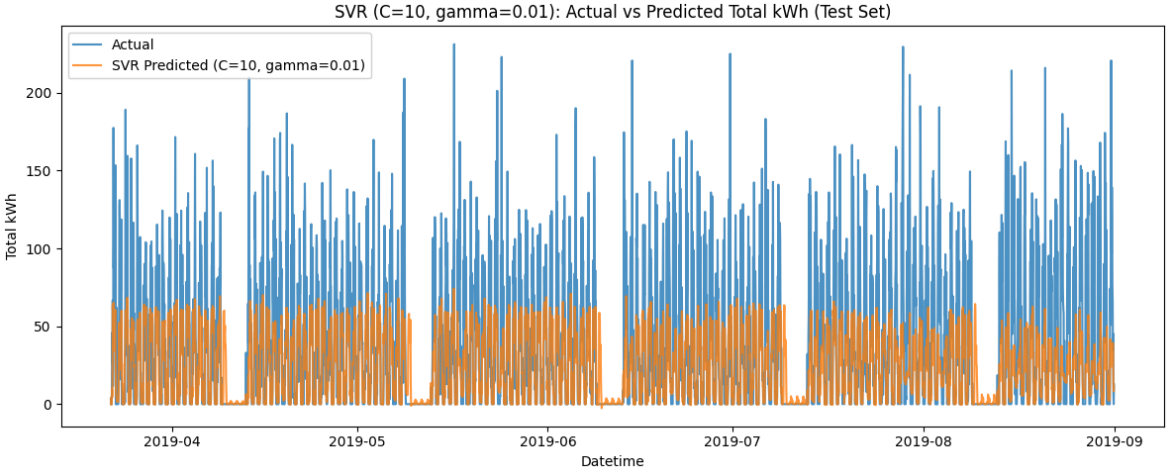
Implementing a Support Vector Regression (SVR) Model

I am now training a SVR model, which is a powerful algorithm capable of capturing non-linear relationships in the data. I have configured the model with a radial basis function (RBF) kernel and specific hyperparameters that I have tuned for this problem. After fitting the model on the training data, I use it to generate predictions for the validation and test sets. I then evaluate its performance using my standard set of metrics and create a final visualisation that plots the model's predictions against the actual test data, saving the chart for my dissertation.

In [90]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Instantiate SVR with your chosen parameters
svr = SVR(kernel='rbf', C=10, gamma= 0.01, epsilon=0.1)

# Fit on training data
svr.fit(X_train, y_train)

# Predict on validation and test sets
y_val_pred_svr = svr.predict(X_val)
y_test_pred_svr = svr.predict(X_test)

# Evaluate performance
print_metrics(y_val, y_val_pred_svr, 'SVR Validation (C=10, gamma=1)')
```

```
print_metrics(y_test, y_test_pred_svr, 'SVR Test (C=10, gamma=1)')

# Plot Actual vs Predicted
import matplotlib.pyplot as plt
plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test, label='Actual', alpha=0.8)
plt.plot(y_test.index, y_test_pred_svr, label='SVR Predicted (C=10, gamma=0.01)'
plt.legend()
plt.title('SVR (C=10, gamma=0.01): Actual vs Predicted Total kWh (Test Set)')
plt.xlabel('Datetime')
plt.ylabel('Total kWh')
plt.tight_layout()
plt.savefig("SVR_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```

```
SVR Validation (C=10, gamma=1) MAE: 14.97
SVR Validation (C=10, gamma=1) RMSE: 26.41
SVR Validation (C=10, gamma=1) R^2: 0.553
SVR Test (C=10, gamma=1) MAE: 21.55
SVR Test (C=10, gamma=1) RMSE: 36.44
SVR Test (C=10, gamma=1) R^2: 0.308
```



"With proper scaling and kernel parameter selection, SVR was able to capture the periodic trends and moderate fluctuations in hourly EV charging demand. However, it still underpredicts the most extreme demand peaks, a limitation typical of SVR when forecasting highly volatile, event-driven time series. These results support the need for more flexible models, such as tree-based ensembles or deep neural networks, to better capture rare but impactful spikes in demand."

Install Updated XGBoost

In [91]:
```
!pip install --upgrade xgboost
```

```
Collecting xgboost
  Downloading xgboost-3.0.4-py3-none-manylinux_2_28_x86_64.whl.metadata (2.1 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from xgboost) (2.0.2)
Collecting nvidia-nccl-cu12 (from xgboost)
  Downloading nvidia_nccl_cu12-2.27.7-py3-none-manylinux2014_x86_64.manylinux_2_1
7_x86_64.whl.metadata (2.0 kB)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from xgboost) (1.16.1)
Downloading xgboost-3.0.4-py3-none-manylinux_2_28_x86_64.whl (94.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 94.9/94.9 MB 12.6 MB/s eta 0:00:00
Downloading nvidia_nccl_cu12-2.27.7-py3-none-manylinux2014_x86_64.manylinux_2_17_
x86_64.whl (322.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.5/322.5 MB 3.2 MB/s eta 0:00:00
Installing collected packages: nvidia-nccl-cu12, xgboost
Successfully installed nvidia-nccl-cu12-2.27.7 xgboost-3.0.4
```

In [92]:
```
import xgboost as xgb
print(xgb.__version__)
```

```
3.0.4
```

Optimising the XGBoost Model

To find the best possible configuration for my XGBoost model, I am performing hyperparameter tuning using a randomised search. I have defined a range of potential settings for key parameters like the number of trees, their maximum depth, and the learning rate. The RandomizedSearchCV function will then automatically test 20 different combinations of these settings, using 3-fold cross-validation to evaluate each one robustly. This process automates the search for the most effective model configuration, and at the end, it will report the set of parameters that yielded the best performance on my training data.

In [94]:
```
from sklearn.model_selection import RandomizedSearchCV
import xgboost as xgb

params = {
    'n_estimators': [100, 250, 500],
    'max_depth': [3, 5, 7, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 1.0],
    'colsample_bytree': [0.7, 0.8, 1.0],
    'min_child_weight': [1, 3, 5],
    'gamma': [0, 0.1, 1]
}

xgb_model = xgb.XGBRegressor(objective='reg:squarederror', random_state=42)
random_search = RandomizedSearchCV(
    xgb_model, params,
    scoring='neg_mean_squared_error',
    cv=3,
    n_iter=20,
    verbose=1,
    n_jobs=-1
)
```

```
random_search.fit(X_train, y_train)
print("Best parameters:", random_search.best_params_)

print("Best parameters:", random_search.best_params_)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Best parameters: {'subsample': 0.8, 'n_estimators': 500, 'min_child_weight': 3,
'max_depth': 5, 'learning_rate': 0.01, 'gamma': 0.1, 'colsample_bytree': 0.8}
Best parameters: {'subsample': 0.8, 'n_estimators': 500, 'min_child_weight': 3,
'max_depth': 5, 'learning_rate': 0.01, 'gamma': 0.1, 'colsample_bytree': 0.8}
```

Evaluating the Optimised Model

Now that the hyperparameter search has identified the best configuration for the XGBoost model, I am using that optimal version to make final predictions. I first refit this best estimator on the full training set. Then, I generate forecasts for both the validation and test datasets. Finally, I calculate my standard performance metrics—MAE, RMSE, and R-squared—to get a definitive measure of how accurately the fully optimised model performs on unseen data.

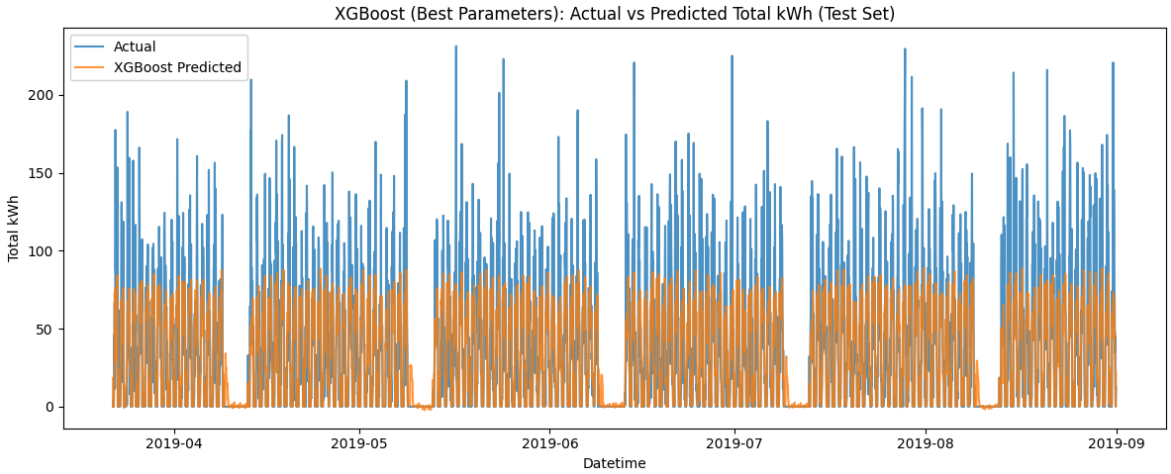
In [95]:
```
# Refit with Best Parameters and Predict

best_xgb = random_search.best_estimator_
y_val_pred_xgb = best_xgb.predict(X_val)
y_test_pred_xgb = best_xgb.predict(X_test)

def print_metrics(y_true, y_pred, set_name='Set'):
    from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred) ** 0.5
    r2 = r2_score(y_true, y_pred)
    print(f"{set_name} MAE: {mae:.2f}")
    print(f"{set_name} RMSE: {rmse:.2f}")
    print(f"{set_name} R^2: {r2:.3f}")

print_metrics(y_val, y_val_pred_xgb, 'XGBoost Validation')
print_metrics(y_test, y_test_pred_xgb, 'XGBoost Test')
```

```
XGBoost Validation MAE: 12.99
XGBoost Validation RMSE: 22.86
XGBoost Validation R^2: 0.665
XGBoost Test MAE: 15.90
XGBoost Test RMSE: 27.43
XGBoost Test R^2: 0.608
```

Visualising the Optimised Model's Performance

To provide a final, intuitive assessment of the optimised XGBoost model, I am plotting its predictions against the actual values from the test set. This graph directly compares the forecast energy demand with the real-world data, offering a clear visual representation of the model's accuracy. I have customised the plot with a title and labels for clarity before saving the final visualisation as a high-resolution image for my dissertation.

In [96]:
```
# Plot Actual vs Predicted
import matplotlib.pyplot as plt

plt.figure(figsize=(12,5))
plt.plot(y_test.index, y_test, label='Actual', alpha=0.8)
plt.plot(y_test.index, y_test_pred_xgb, label='XGBoost Predicted', alpha=0.8)
plt.legend()
plt.title('XGBoost (Best Parameters): Actual vs Predicted Total kWh (Test Set)')
plt.xlabel('Datetime')
plt.ylabel('Total kWh')
plt.tight_layout()
plt.savefig("XGBoost_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Training and Optimising a Random Forest Model

I am now training a Random Forest model, which is another powerful ensemble learning method. To ensure the model is as accurate as possible, I first define a grid of potential hyperparameter settings. I then use a randomised search with cross-validation to automatically test 20 different combinations of these settings, which efficiently identifies the best-performing configuration. Once the optimal parameters are found, I use that best model to make predictions on my validation and test sets and then evaluate its performance using my standard set of metrics.

In [97]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# --- 1. Define Hyperparameter Grid for Random Forest ---
rf_params = {
    'n_estimators': [100, 250, 500],
    'max_depth': [5, 10, 15, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [1.0, 'sqrt', 'log2']
}

# --- 2. Initialize the Model and Randomized Search ---
# Note: RandomForestRegressor is imported from sklearn.ensemble
```

```
rf_model = RandomForestRegressor(random_state=42)

random_search_rf = RandomizedSearchCV(
    rf_model, rf_params,
    scoring='neg_mean_squared_error',
    cv=3,
    n_iter=20, # Same number of iterations for a fair comparison
    verbose=1,
    n_jobs=-1
)

# --- 3. Fit, Predict, and Evaluate ---
random_search_rf.fit(X_train, y_train)
print("Best Random Forest parameters:", random_search_rf.best_params_)

best_rf = random_search_rf.best_estimator_
y_val_pred_rf = best_rf.predict(X_val)
y_test_pred_rf = best_rf.predict(X_test)

def print_metrics(y_true, y_pred, set_name='Set'):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred) ** 0.5
    r2 = r2_score(y_true, y_pred)
    print(f"{set_name} MAE: {mae:.2f}")
    print(f"{set_name} RMSE: {rmse:.2f}")
    print(f"{set_name} R^2: {r2:.3f}")

print("\n--- Random Forest Results ---")
print_metrics(y_val, y_val_pred_rf, 'Random Forest Validation')
print_metrics(y_test, y_test_pred_rf, 'Random Forest Test')
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
Best Random Forest parameters: {'n_estimators': 250, 'min_samples_split': 10, 'mi
n_samples_leaf': 2, 'max_features': 1.0, 'max_depth': 10}

--- Random Forest Results ---
Random Forest Validation MAE: 12.99
Random Forest Validation RMSE: 23.11
Random Forest Validation R^2: 0.657
Random Forest Test MAE: 15.94
Random Forest Test RMSE: 27.19
Random Forest Test R^2: 0.615
```

In [98]:
```
!pip install lightgbm
```

```
Collecting lightgbm
  Downloading lightgbm-4.6.0-py3-none-manylinux_2_28_x86_64.whl.metadata (17 kB)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.11/dist-pa
ckages (from lightgbm) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from lightgbm) (1.16.1)
Downloading lightgbm-4.6.0-py3-none-manylinux_2_28_x86_64.whl (3.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/3.6 MB ? eta -:--:--
   ━━━━ ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.3/3.6 MB 9.9 MB/s eta 0:00:01
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ ━━━━ 3.0/3.6 MB 45.2 MB/s eta 0:00:01
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 38.8 MB/s eta 0:00:00
Installing collected packages: lightgbm
Successfully installed lightgbm-4.6.0
```

In [99]:
```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.4.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from optuna) (25.0)
Collecting sqlalchemy>=1.4.2 (from optuna)
  Downloading sqlalchemy-2.0.43-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (9.6 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (f
rom optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages
(from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from alemb
ic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.
11/dist-packages (from alembic>=1.5.0->optuna) (4.14.1)
Collecting greenlet>=1 (from sqlalchemy>=1.4.2->optuna)
  Downloading greenlet-3.2.4-cp311-cp311-manylinux_2_24_x86_64.manylinux_2_28_x86
_64.whl.metadata (4.1 kB)
Downloading optuna-4.4.0-py3-none-any.whl (395 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 395.9/395.9 kB 8.7 MB/s eta 0:00:00
Downloading alembic-1.16.4-py3-none-any.whl (247 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.0/247.0 kB 20.1 MB/s eta 0:00:00
Downloading sqlalchemy-2.0.43-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (3.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 57.6 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Downloading greenlet-3.2.4-cp311-cp311-manylinux_2_24_x86_64.manylinux_2_28_x86_6
4.whl (587 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 587.7/587.7 kB 37.9 MB/s eta 0:00:00
Installing collected packages: greenlet, colorlog, sqlalchemy, alembic, optuna
Successfully installed alembic-1.16.4 colorlog-6.9.0 greenlet-3.2.4 optuna-4.4.0
sqlalchemy-2.0.43
```

Optimising the LightGBM Model with Optuna

I am now using a more advanced optimisation framework called Optuna to find the best possible hyperparameters for a LightGBM model. I first define an 'objective' function that Optuna will try to minimise; this function trains a LightGBM model with a specific set of parameters and evaluates its performance on the validation set. Optuna then intelligently runs 20 trials, each time selecting a new combination of parameters to try and achieve the lowest possible error. Once this automated search is complete, I retrain a final model using the best-performing parameters on the combined training and validation data. Finally, I evaluate this fully optimised model on the test set to get a definitive measure of its performance.

In [74]:
```
import lightgbm as lgb
import optuna
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

```
# Silence Optuna's trial-by-trial logging ---
optuna.logging.set_verbosity(optuna.logging.WARNING)

# Define the Objective Function for Optuna with Early Stopping ---
def objective_lgbm(trial):
    """Defines the LightGBM training process for a single Optuna trial."""

    params = {
        'objective': 'regression_l1',
        'metric': 'rmse',
        'n_estimators': trial.suggest_int('n_estimators', 400, 2000),
        'learning_rate': trial.suggest_float('learning_rate', 1e-3, 0.2, log=Tru
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'num_leaves': trial.suggest_int('num_leaves', 20, 50),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0),
        'random_state': 42,
        'n_jobs': -1
    }

    # --- MODIFIED: Added verbose=-1 to silence LightGBM's internal messages ---
    model = lgb.LGBMRegressor(**params, verbose=-1)

    model.fit(
        X_train, y_train,
        eval_set=[(X_val, y_val)],
        eval_metric='rmse',
        callbacks=[lgb.early_stopping(15, verbose=False)]
    )

    preds = model.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, preds))

    return rmse

# --- 2. Create and Run the Optuna Study ---
print("--- Starting LightGBM Hyperparameter Optimization with Optuna ---")
study_lgbm = optuna.create_study(direction='minimize')
study_lgbm.optimize(objective_lgbm, n_trials=20)

# --- 3. Display the Best Results ---
print("\n--- Optimization Complete ---")
print("Best LightGBM trial:")
best_trial_lgbm = study_lgbm.best_trial
print(f"  Value (Validation RMSE): {best_trial_lgbm.value:.4f}")
print("  Params: ")
for key, value in best_trial_lgbm.params.items():
    print(f"    {key}: {value}")

# --- 4. Train Final Model with Best Parameters ---
print("\n--- Training Final Model with Optimal Parameters ---")
final_lgbm_model = lgb.LGBMRegressor(**best_trial_lgbm.params, random_state=42,

X_train_val = pd.concat([X_train, X_val])
y_train_val = pd.concat([y_train, y_val])
final_lgbm_model.fit(X_train_val, y_train_val)

# --- 5. Evaluate on Validation and Test Sets ---
y_val_pred_lgbm = final_lgbm_model.predict(X_val)
```

```
y_test_pred_lgbm = final_lgbm_model.predict(X_test)

val_mae = mean_absolute_error(y_val, y_val_pred_lgbm)
val_rmse = np.sqrt(mean_squared_error(y_val, y_val_pred_lgbm))
val_r2 = r2_score(y_val, y_val_pred_lgbm)

test_mae = mean_absolute_error(y_test, y_test_pred_lgbm)
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_lgbm))
test_r2 = r2_score(y_test, y_test_pred_lgbm)

print("\n--- Final Optimized LightGBM Performance on Validation Set ---")
print(f"MAE: {val_mae:.4f}")
print(f"RMSE: {val_rmse:.4f}")
print(f"R^2:  {val_r2:.4f}")

print("\n--- Final Optimized LightGBM Performance on Test Set ---")
print(f"MAE: {test_mae:.4f}")
print(f"RMSE: {test_rmse:.4f}")
print(f"R^2:  {test_r2:.4f}")
```

```
--- Starting LightGBM Hyperparameter Optimization with Optuna ---

--- Optimization Complete ---
Best LightGBM trial:
  Value (Validation RMSE): 23.3039
  Params:
    n_estimators: 436
    learning_rate: 0.17272767077968096
    max_depth: 9
    num_leaves: 21
    subsample: 0.8576378648009729
    colsample_bytree: 0.9882645753785287

--- Training Final Model with Optimal Parameters ---

--- Final Optimized LightGBM Performance on Validation Set ---
MAE: 8.0583
RMSE: 13.7428
R^2:  0.8788

--- Final Optimized LightGBM Performance on Test Set ---
MAE: 17.1303
RMSE: 28.3673
R^2:  0.5807
```

CNN-LSTM for Time Series Forecasting

Preparing Data for Sequential Models

I am now restructuring the data into overlapping sequences, a crucial step for preparing it for time-series forecasting with neural networks. I have created a function that transforms the flat data into a 3D format by creating a sliding window. Each sequence contains 24 hours of historical data, which will be used to predict the single hour that follows it. Finally, I confirm the new 3D shape of my training, validation, and test sets to ensure they are correctly formatted for the model.

In [103…

```
import numpy as np

# Define the Sequencing Function for NumPy arrays ---
# This function converts your 2D data (samples, features) into
# 3D sequences (samples, window_size, features).

def create_sequences_np(X_data, y_data, window_size=24):
    """
    Creates 3D sequences from 2D data.
    Works with NumPy arrays.
    """
    Xs, ys = [], []
    for i in range(len(X_data) - window_size):
        # Standard NumPy slicing is used here
        v = X_data[i:(i + window_size)]
        Xs.append(v)
        ys.append(y_data[i + window_size])
    return np.array(Xs), np.array(ys)


# --- Step 2: Create the Sequenced Data ---
# Use the final imputed and scaled NumPy arrays from the previous steps.
# (X_train_scaled, y_train_scaled, etc.)

window_size = 24 # e.g., use 24 hours of data to predict the next hour

print("Creating sequences from scaled data...")

# Note: We use .flatten() on the y arrays to make them 1D, which is what the fun
X_train_seq, y_train_seq = create_sequences_np(X_train_scaled, y_train_scaled.fl
X_val_seq, y_val_seq = create_sequences_np(X_val_scaled, y_val_scaled.flatten(),
X_test_seq, y_test_seq = create_sequences_np(X_test_scaled, y_test_scaled.flatte


# --- Step 3: Verify the Shapes of the New Variables ---
# The X shapes should be 3D and the y shapes should be 1D.
print(f"Shape of X_train_seq: {X_train_seq.shape}")
print(f"Shape of y_train_seq: {y_train_seq.shape}")
print(f"Shape of X_val_seq: {X_val_seq.shape}")
print(f"Shape of y_val_seq: {y_val_seq.shape}")

print("\nData has been successfully converted into sequences.")
```

```
Creating sequences from scaled data...
Shape of X_train_seq: (18215, 24, 14)
Shape of y_train_seq: (18215,)
Shape of X_val_seq: (3884, 24, 14)
Shape of y_val_seq: (3884,)

Data has been successfully converted into sequences.
```

In [100…

```
!pip install tensorflow
```

```
Collecting tensorflow
  Downloading tensorflow-2.20.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x
86_64.whl.metadata (4.5 kB)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-p
ackages (from tensorflow) (1.4.0)
Collecting astunparse>=1.6.0 (from tensorflow)
  Downloading astunparse-1.6.3-py2.py3-none-any.whl.metadata (4.4 kB)
Collecting flatbuffers>=24.3.25 (from tensorflow)
  Downloading flatbuffers-25.2.10-py2.py3-none-any.whl.metadata (875 bytes)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/
lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Collecting google_pasta>=0.1.1 (from tensorflow)
  Downloading google_pasta-0.2.0-py3-none-any.whl.metadata (814 bytes)
Collecting libclang>=13.0.0 (from tensorflow)
  Downloading libclang-18.1.1-py2.py3-none-manylinux2010_x86_64.whl.metadata (5.2
kB)
Requirement already satisfied: opt_einsum>=2.3.2 in /usr/local/lib/python3.11/dis
t-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packag
es (from tensorflow) (25.0)
Requirement already satisfied: protobuf>=5.28.0 in /usr/local/lib/python3.11/dist
-packages (from tensorflow) (6.31.1)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.11/d
ist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packa
ges (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-pack
ages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.11/dist
-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing_extensions>=3.6.6 in /usr/local/lib/python
3.11/dist-packages (from tensorflow) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-pa
ckages (from tensorflow) (1.17.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.11/d
ist-packages (from tensorflow) (1.74.0)
Collecting tensorboard~=2.20.0 (from tensorflow)
  Downloading tensorboard-2.20.0-py3-none-any.whl.metadata (1.8 kB)
Requirement already satisfied: keras>=3.10.0 in /usr/local/lib/python3.11/dist-pa
ckages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy>=1.26.0 in /usr/local/lib/python3.11/dist-pa
ckages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-pac
kages (from tensorflow) (3.14.0)
Requirement already satisfied: ml_dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.
11/dist-packages (from tensorflow) (0.5.3)
Collecting wheel<1.0,>=0.23.0 (from astunparse>=1.6.0->tensorflow)
  Downloading wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (f
rom keras>=3.10.0->tensorflow) (14.1.0)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras>=3.10.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras>=3.10.0->tensorflow) (0.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
```

```
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests<3,>=2.21.0->tensorflow) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/lib/python3/dist-packages
(from tensorboard~=2.20.0->tensorflow) (3.3.6)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages
(from tensorboard~=2.20.0->tensorflow) (11.3.0)
Collecting tensorboard-data-server<0.8.0,>=0.7.0 (from tensorboard~=2.20.0->tenso
rflow)
  Downloading tensorboard_data_server-0.7.2-py3-none-manylinux_2_31_x86_64.whl.me
tadata (1.1 kB)
Collecting werkzeug>=1.0.1 (from tensorboard~=2.20.0->tensorflow)
  Downloading werkzeug-3.1.3-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.11/dis
t-packages (from werkzeug>=1.0.1->tensorboard~=2.20.0->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.1
1/dist-packages (from rich->keras>=3.10.0->tensorflow) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.
11/dist-packages (from rich->keras>=3.10.0->tensorflow) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packa
ges (from markdown-it-py>=2.2.0->rich->keras>=3.10.0->tensorflow) (0.1.2)
Downloading tensorflow-2.20.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (620.6 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 620.6/620.6 MB 1.7 MB/s eta 0:00:00
Downloading astunparse-1.6.3-py2.py3-none-any.whl (12 kB)
Downloading flatbuffers-25.2.10-py2.py3-none-any.whl (30 kB)
Downloading google_pasta-0.2.0-py3-none-any.whl (57 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 57.5/57.5 kB 5.3 MB/s eta 0:00:00
Downloading libclang-18.1.1-py2.py3-none-manylinux2010_x86_64.whl (24.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.5/24.5 MB 84.2 MB/s eta 0:00:00
Downloading tensorboard-2.20.0-py3-none-any.whl (5.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.5/5.5 MB 104.7 MB/s eta 0:00:00
Downloading tensorboard_data_server-0.7.2-py3-none-manylinux_2_31_x86_64.whl (6.6
MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.6/6.6 MB 100.0 MB/s eta 0:00:00
Downloading werkzeug-3.1.3-py3-none-any.whl (224 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 224.5/224.5 kB 18.4 MB/s eta 0:00:00
Downloading wheel-0.45.1-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.5/72.5 kB 7.0 MB/s eta 0:00:00
Installing collected packages: libclang, flatbuffers, wheel, werkzeug, tensorboar
d-data-server, google_pasta, tensorboard, astunparse, tensorflow
Successfully installed astunparse-1.6.3 flatbuffers-25.2.10 google_pasta-0.2.0 li
bclang-18.1.1 tensorboard-2.20.0 tensorboard-data-server-0.7.2 tensorflow-2.20.0
werkzeug-3.1.3 wheel-0.45.1
```

Building and Training a CNN-LSTM Network

I am now constructing a hybrid neural network designed specifically for time-series forecasting. The architecture begins with a convolutional layer (Conv1D) to efficiently extract key features from the input sequences, followed by an LSTM layer which is adept at learning long-range dependencies and temporal patterns. To ensure the model trains effectively and avoids memorising the training data, a phenomenon known as overfitting, I have incorporated regularisation techniques such as BatchNormalization and Dropout.

After defining the architecture, I compile the model, specifying the 'Adam' optimiser to guide its learning process and 'Mean Squared Error' as the loss function to measure its accuracy. The model is then trained on the prepared sequential data. I have also

implemented an 'Early Stopping' mechanism, which automatically halts the training if performance on the validation set ceases to improve, ensuring an efficient process and preventing overfitting. Finally, I plot the training and validation loss to visually inspect the model's learning progress and confirm that it is generalising well to new data.

In [104…

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, BatchNormaliza
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Get the input shape from your sequenced training data
# This assumes you have already run the create_sequences function on your
# final imputed and scaled data.
window_size = X_train_seq.shape[1]
n_features = X_train_seq.shape[2]


# Define a Simpler, More Robust CNN-LSTM Architecture ---
print("Building a simplified CNN-LSTM model to combat overfitting...")

model_v2 = Sequential([
    # A single Conv1D layer is often enough to extract features from sequences.
    Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(window_siz
    BatchNormalization(),
    Dropout(0.3),

    # A single LSTM layer. Note the 'tanh' activation, which is standard.
    # return_sequences=False because it's the last recurrent layer before the De
    LSTM(50, activation='tanh'),
    Dropout(0.4), # A slightly higher dropout for better regularization

    # A smaller Dense layer before the final output.
    Dense(25, activation='relu'),

    # Final output neuron for regression.
    Dense(1)
])


# Compile the Model ---
# We'll use a slightly lower learning rate for more stable training.
optimizer = tf.keras.optimizers.Adam(learning_rate=0.0005)
model_v2.compile(optimizer=optimizer, loss='mse') # Mean Squared Error for regre
model_v2.summary()


# Set Up Early Stopping ---
# This prevents the model from training for too long once it stops improving.
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights


# Train the Model ---
history_v2 = model_v2.fit(
    X_train_seq, y_train_seq,
    epochs=100, # Early stopping will likely finish before 100 epochs
    batch_size=32,
    validation_data=(X_val_seq, y_val_seq),
```

```
    callbacks=[early_stop],
    verbose=1
)


# Plot Training and Validation Loss ---
# This plot is the most important indicator of whether we've reduced overfitting
plt.figure(figsize=(10, 6))
plt.plot(history_v2.history['loss'], label='Training Loss')
plt.plot(history_v2.history['val_loss'], label='Validation Loss')
plt.title('Revised CNN-LSTM Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.savefig("CNN_LSTM_Loss_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```

```
Building a simplified CNN-LSTM model to combat overfitting...
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 22, 64)	2,752
batch_normalization (BatchNormalization)	(None, 22, 64)	256
dropout (Dropout)	(None, 22, 64)	0
lstm (LSTM)	(None, 50)	23,000
dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 25)	1,275
dense_1 (Dense)	(None, 1)	26

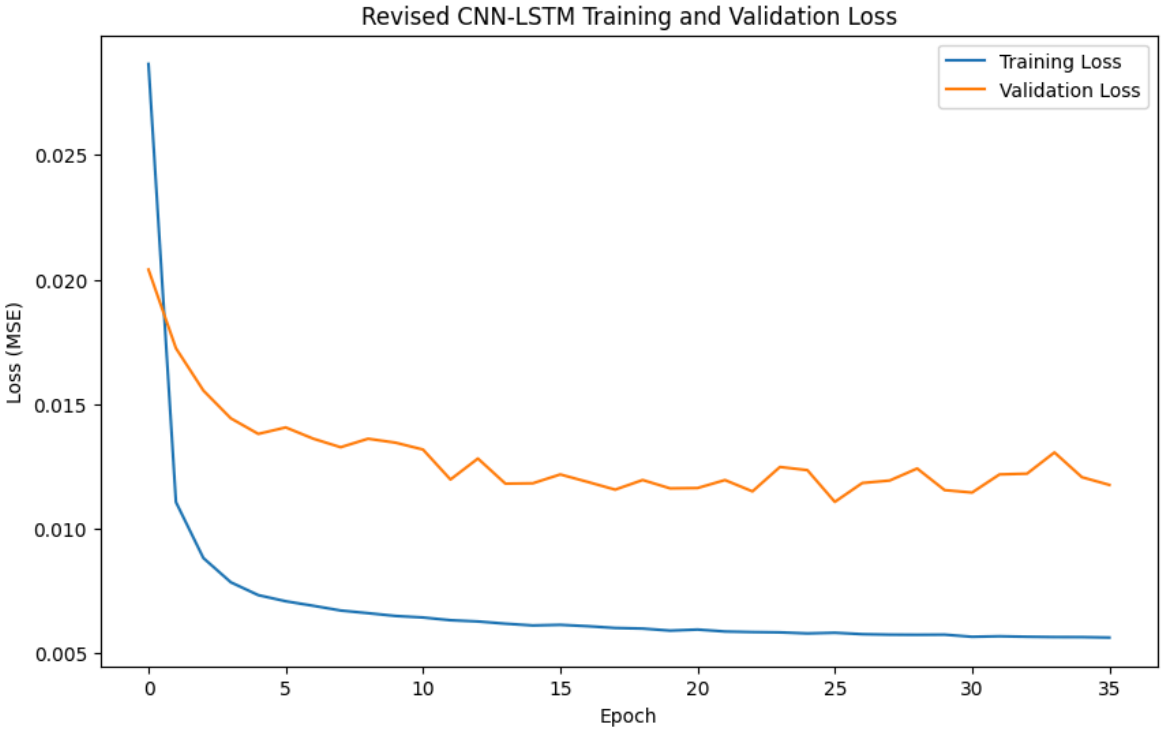
Total params: 27,309 (106.68 KB)

Trainable params: 27,181 (106.18 KB)

Non-trainable params: 128 (512.00 B)

```
Epoch 1/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 8s 10ms/step - loss: 0.0495 - val_loss: 0.0204
Epoch 2/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.0121 - val_loss: 0.0172
Epoch 3/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0095 - val_loss: 0.0155
Epoch 4/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - loss: 0.0080 - val_loss: 0.0144
Epoch 5/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0074 - val_loss: 0.0138
Epoch 6/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0071 - val_loss: 0.0141
Epoch 7/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0071 - val_loss: 0.0136
Epoch 8/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0069 - val_loss: 0.0133
Epoch 9/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0067 - val_loss: 0.0136
Epoch 10/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0064 - val_loss: 0.0135
Epoch 11/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0064 - val_loss: 0.0132
Epoch 12/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0067 - val_loss: 0.0120
Epoch 13/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0062 - val_loss: 0.0128
Epoch 14/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0063 - val_loss: 0.0118
Epoch 15/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0062 - val_loss: 0.0118
Epoch 16/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0061 - val_loss: 0.0122
Epoch 17/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0119
Epoch 18/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0061 - val_loss: 0.0116
Epoch 19/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0060 - val_loss: 0.0120
Epoch 20/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0116
Epoch 21/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0062 - val_loss: 0.0116
Epoch 22/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0120
Epoch 23/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0115
Epoch 24/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0057 - val_loss: 0.0125
Epoch 25/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0058 - val_loss: 0.0124
Epoch 26/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0058 - val_loss: 0.0111
Epoch 27/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0118
Epoch 28/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0058 - val_loss: 0.0119
Epoch 29/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0059 - val_loss: 0.0124
Epoch 30/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0057 - val_loss: 0.0116
```

```
Epoch 31/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0056 - val_loss: 0.0115
Epoch 32/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0057 - val_loss: 0.0122
Epoch 33/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0057 - val_loss: 0.0122
Epoch 34/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0056 - val_loss: 0.0131
Epoch 35/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0055 - val_loss: 0.0121
Epoch 36/100
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 10ms/step - loss: 0.0056 - val_loss: 0.0118
```



Evaluating the Neural Network's Performance

Now that my neural network is trained, I am evaluating its performance on the unseen data. First, I use the trained model to generate predictions on the validation and test sets. Since these predictions are in a normalised format (between 0 and 1), I then reverse the scaling process to convert both the predicted and the actual values back to their original units of kWh. This allows for a meaningful, real-world comparison. Finally, I calculate the Mean Absolute Error, Root Mean Squared Error, and R-squared score to quantitatively measure the model's accuracy on both the validation and test datasets.

In [105…

```
# Predict on the validation and test sequences ---
print("Making predictions on validation and test sets...")
y_val_pred_scaled = model_v2.predict(X_val_seq)
y_test_pred_scaled = model_v2.predict(X_test_seq)


# Inverse scale ALL values to their original units (e.g., kWh) ---
print("Inverse-scaling predictions and true values...")

# Inverse scale the model's predictions
y_val_pred = scaler_y.inverse_transform(y_val_pred_scaled)
```

```
y_test_pred_original_lstm = scaler_y.inverse_transform(y_test_pred_scaled)

# Inverse scale the true values for a fair comparison
# This is the step that creates the missing 'y_val_true' variable
y_val_true = scaler_y.inverse_transform(y_val_seq.reshape(-1, 1))
y_test_true = scaler_y.inverse_transform(y_test_seq.reshape(-1, 1))


# Define metrics function and evaluate performance ---
def print_metrics(y_true, y_pred, set_name='Set'):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)

    print(f"\n--- {set_name} Metrics ---")
    print(f"MAE:  {mae:.2f}")
    print(f"RMSE: {rmse:.2f}")
    print(f"R^2:  {r2:.3f}")

# Print metrics for both validation and test sets
print_metrics(y_val_true, y_val_pred, 'CNN-LSTM Validation')
print_metrics(y_test_true, y_test_pred_original_lstm, 'CNN-LSTM Test')
```

```
Making predictions on validation and test sets...
122/122 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step
Inverse-scaling predictions and true values...

--- CNN-LSTM Validation Metrics ---
MAE:  17.06
RMSE: 25.71
R^2:  0.576

--- CNN-LSTM Test Metrics ---
MAE:  19.30
RMSE: 29.01
R^2:  0.560
```

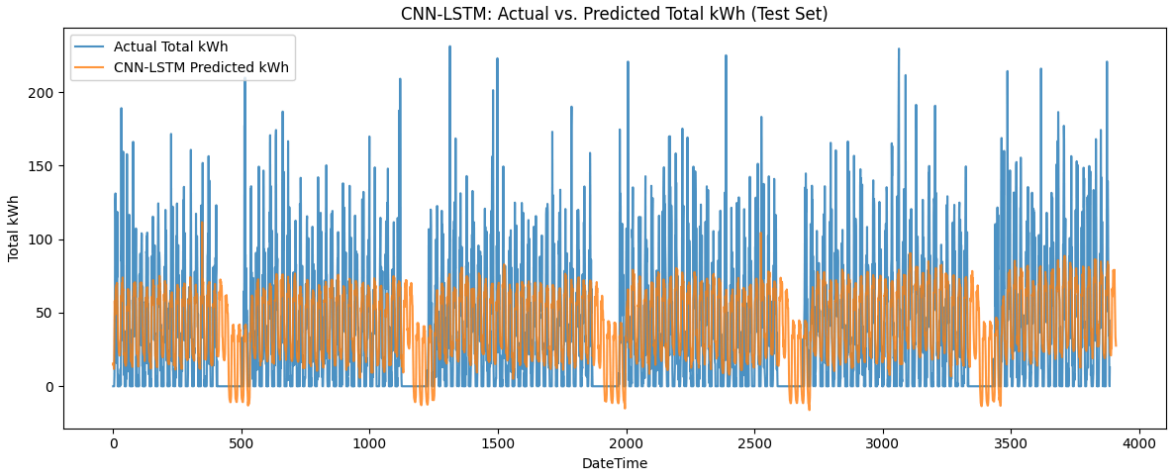
Visualising the Neural Network's Performance

To provide a final, intuitive assessment of my neural network, I am plotting its predictions against the actual values from the test set. This graph directly compares the forecast energy demand with the real-world data, offering a clear visual representation of the model's accuracy on unseen data. I have customised the plot with a title and labels for clarity before saving the final visualisation as a high-resolution image for my dissertation.

In [106…

```
import matplotlib.pyplot as plt

# Plot Actual vs. Predicted for the Test Set ---
plt.figure(figsize=(12, 5))
plt.plot(y_test_true, label='Actual Total kWh', alpha=0.8)
plt.plot(y_test_pred, label='CNN-LSTM Predicted kWh', alpha=0.8)

plt.title('CNN-LSTM: Actual vs. Predicted Total kWh (Test Set)')
plt.xlabel('DateTime')
plt.ylabel('Total kWh')
plt.legend()
```

```
plt.tight_layout()
plt.savefig("CNN-LSTM_visualisation.png", dpi=1200, bbox_inches='tight')
plt.show()
```



Hyperparameter Tuning for CNN-LSTM

In [107…

```
!pip install keras-tuner
```

```
Collecting keras-tuner
  Downloading keras_tuner-1.4.7-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages
(from keras-tuner) (3.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packag
es (from keras-tuner) (25.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-package
s (from keras-tuner) (2.32.3)
Collecting kt-legacy (from keras-tuner)
  Downloading kt_legacy-1.0.5-py3-none-any.whl.metadata (221 bytes)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages (f
rom keras->keras-tuner) (14.1.0)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages (f
rom keras->keras-tuner) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras->keras-tuner) (0.17.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-packag
es (from keras->keras-tuner) (0.5.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests->keras-tuner) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests->keras-tuner) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
st-packages (from requests->keras-tuner) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/di
st-packages (from requests->keras-tuner) (2025.8.3)
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python
3.11/dist-packages (from optree->keras->keras-tuner) (4.14.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.1
1/dist-packages (from rich->keras->keras-tuner) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.
11/dist-packages (from rich->keras->keras-tuner) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packa
ges (from markdown-it-py>=2.2.0->rich->keras->keras-tuner) (0.1.2)
Downloading keras_tuner-1.4.7-py3-none-any.whl (129 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 129.1/129.1 kB 2.7 MB/s eta 0:00:00
Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.4.7 kt-legacy-1.0.5
```

Optimising the Neural Network with Keras Tuner

To ensure my neural network performs as accurately as possible, I am now using the Keras Tuner library to automatically find the best hyperparameter configuration. I first define a function that builds the model but leaves key parameters—such as the number of filters, the size of the LSTM and dense layers, the dropout rates, and the learning rate—as variables for the tuner to adjust.

I then configure a RandomSearch tuner to test 10 different combinations of these parameters, training each one and evaluating its performance on the validation set. This automated process systematically searches for the most effective model architecture. Once the search is complete, I retrieve the best-performing model and print a summary of the optimal hyperparameter values it discovered. This data-driven approach ensures that my final model is not just based on an educated guess, but has been systematically optimised for this specific forecasting task.

In [108…

```
import tensorflow as tf
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout, BatchNormaliza
from tensorflow.keras.optimizers import Adam


# Ensure prerequisite variables exist
# This code assumes the following variables are already defined and populated fr
# X_train_seq, y_train_seq, X_val_seq, y_val_seq

# Define the Model Building Function ---
# This function creates a Keras model but uses the 'hp' object to define
# hyperparameters that we want to tune. Keras Tuner will call this function
# repeatedly with different hyperparameter values.

def build_model(hp):
    """Builds a CNN-LSTM model for hyperparameter tuning."""
    model = Sequential()

    # Get the input shape from the training data
    window_size = X_train_seq.shape[1]
    n_features = X_train_seq.shape[2]

    # Tune the Conv1D Layer ---
    hp_filters = hp.Int('filters', min_value=32, max_value=128, step=32)
    model.add(Conv1D(
        filters=hp_filters,
        kernel_size=3,
        activation='relu',
        input_shape=(window_size, n_features)
    ))
    model.add(BatchNormalization())

    # Tune the Dropout Rate after Conv1D ---
    hp_conv_dropout = hp.Float('conv_dropout', min_value=0.1, max_value=0.5, ste
    model.add(Dropout(hp_conv_dropout))

    # Tune the LSTM Layer ---
    hp_lstm_units = hp.Int('lstm_units', min_value=30, max_value=100, step=20)
    model.add(LSTM(
        units=hp_lstm_units,
        activation='tanh' # Using tanh is standard for LSTMs
    ))

    # Tune the Dropout Rate after LSTM ---
    hp_lstm_dropout = hp.Float('lstm_dropout', min_value=0.2, max_value=0.6, ste
    model.add(Dropout(hp_lstm_dropout))
```

```
    # Tune the Dense Layer ---
    hp_dense_units = hp.Int('dense_units', min_value=20, max_value=80, step=20)
    model.add(Dense(units=hp_dense_units, activation='relu'))

    # Output Layer
    model.add(Dense(1))

    # Tune the Learning Rate for the Optimizer ---
    hp_learning_rate = hp.Choice('learning_rate', values=[1e-3, 5e-4, 1e-4])

    model.compile(
        optimizer=Adam(learning_rate=hp_learning_rate),
        loss='mean_squared_error'
    )

    return model

# Instantiate the Tuner ---
# We will use RandomSearch, which randomly tries different combinations.
# 'objective' tells the tuner what to minimize (validation loss).
# 'max_trials' is the number of different models to test.
tuner = kt.RandomSearch(
    build_model,
    objective='val_loss',
    max_trials=10,  # Number of model configurations to test
    executions_per_trial=1, # Number of times to train each model
    directory='keras_tuner_dir',
    project_name='ev_demand_forecasting'
)

# You can clear previous results with the following line (optional)
# tuner.reload()

# Run the Hyperparameter Search ---
# This is the main training step. It will take some time as it trains multiple m
print("Starting hyperparameter search...")
tuner.search(
    X_train_seq, y_train_seq,
    epochs=20, # Use a smaller number of epochs for the search
    validation_data=(X_val_seq, y_val_seq)
)
print("Hyperparameter search complete.")


# Get the Best Model ---
# Retrieve the best model found by the tuner.
best_model = tuner.get_best_models(num_models=1)[0]

# Display the Results ---
print("\n--- Best Model Found ---")
best_model.summary()

# You can also get the best hyperparameters found
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]
print("\n--- Best Hyperparameters ---")
print(f"Filters: {best_hyperparameters.get('filters')}")
print(f"Conv Dropout: {best_hyperparameters.get('conv_dropout'):.2f}")
print(f"LSTM Units: {best_hyperparameters.get('lstm_units')}")
print(f"LSTM Dropout: {best_hyperparameters.get('lstm_dropout'):.2f}")
```

```
print(f"Dense Units: {best_hyperparameters.get('dense_units')}")
print(f"Learning Rate: {best_hyperparameters.get('learning_rate')}")
```

```
Trial 10 Complete [00h 02m 04s]
val_loss: 0.012568959034979343

Best val_loss So Far: 0.01102490071207285
Total elapsed time: 00h 23m 32s
Hyperparameter search complete.

--- Best Model Found ---
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 22, 128)	5,504
batch_normalization (BatchNormalization)	(None, 22, 128)	512
dropout (Dropout)	(None, 22, 128)	0
lstm (LSTM)	(None, 50)	35,800
dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 20)	1,020
dense_1 (Dense)	(None, 1)	21

Total params: 42,857 (167.41 KB)

Trainable params: 42,601 (166.41 KB)

Non-trainable params: 256 (1.00 KB)

```
--- Best Hyperparameters ---
Filters: 128
Conv Dropout: 0.10
LSTM Units: 50
LSTM Dropout: 0.20
Dense Units: 20
Learning Rate: 0.001
```

Final Training and Evaluation of the Optimised Neural Network

Now that the tuner has identified the best architecture, I am taking that optimal model and training it more thoroughly. I train it on my data for a higher number of epochs, using an early stopping mechanism to ensure it stops once its performance on the validation set no longer improves.

After this final training is complete, I use the model to make predictions on the test set. I then convert these predictions from their scaled format back into their original kWh units, which allows for a direct, real-world comparison. Finally, I calculate the definitive

performance metrics for this fully optimised model and generate a plot to visually assess its accuracy on the unseen test data, saving the chart for my dissertation.

In [109…

```
# Retrieve the Best Model Architecture ---
# Get the single best model configuration found by the tuner.
best_model = tuner.get_best_models(num_models=1)[0]
print("--- Best Model Summary ---")
best_model.summary()


# Train the Best Model Properly ---
# We train the best model for more epochs to allow it to fully converge.
# EarlyStopping will monitor the validation loss and stop when it no longer impr
print("\nTraining the best model for more epochs...")
early_stop = EarlyStopping(monitor='val_loss', patience=15, restore_best_weights

history_best = best_model.fit(
    X_train_seq, y_train_seq,
    epochs=150,  # A higher number of epochs
    batch_size=32,
    validation_data=(X_val_seq, y_val_seq),
    callbacks=[early_stop]
)


# Make Predictions ---
print("\nMaking predictions on validation and test sets...")
y_val_pred_scaled = best_model.predict(X_val_seq)
y_test_pred_scaled = best_model.predict(X_test_seq)


# Inverse-Scale Predictions and True Values ---
# Convert the scaled data back to its original units (e.g., kWh) for interpretat
print("Inverse-scaling the data to original units...")
y_val_pred = scaler_y.inverse_transform(y_val_pred_scaled)
y_test_pred = scaler_y.inverse_transform(y_test_pred_scaled)

y_val_true = scaler_y.inverse_transform(y_val_seq.reshape(-1, 1))
y_test_true = scaler_y.inverse_transform(y_test_seq.reshape(-1, 1))


# Evaluate Performance Metrics ---
def print_metrics(y_true, y_pred, set_name='Set'):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r2 = r2_score(y_true, y_pred)

    print(f"\n--- {set_name} Metrics ---")
    print(f"MAE:  {mae:.2f}")
    print(f"RMSE: {rmse:.2f}")
    print(f"R^2:  {r2:.3f}")

# Print metrics for both validation and test sets
print_metrics(y_val_true, y_val_pred, 'Tuned CNN-LSTM Validation')
print_metrics(y_test_true, y_test_pred, 'Tuned CNN-LSTM Test')


# Plot Actual vs. Predicted for the Test Set ---
plt.figure(figsize=(12, 5))
```

```
plt.plot(y_test_true, label='Actual Total kWh', alpha=0.8)
plt.plot(y_test_pred, label='Tuned CNN-LSTM Predicted kWh', alpha=0.8)

plt.title('Tuned CNN-LSTM: Actual vs. Predicted Total kWh (Test Set)')
plt.xlabel('DateTime')
plt.ylabel('Total kWh')
plt.legend()
plt.tight_layout()
plt.savefig("CNN-LSTM Tuned.png", dpi=1200, bbox_inches='tight')
plt.show()


# Plot Training & Validation Loss
# This plot shows how the model learned over the epochs.
plt.figure(figsize=(12, 6))
plt.plot(history_best.history['loss'], label='Training Loss')
plt.plot(history_best.history['val_loss'], label='Validation Loss')
plt.title('Tuned Model Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.savefig("CNN-LSTM Tuned Loss.png", dpi=1200, bbox_inches='tight')
plt.show()
```

```
--- Best Model Summary ---
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 22, 128)	5,504
batch_normalization (BatchNormalization)	(None, 22, 128)	512
dropout (Dropout)	(None, 22, 128)	0
lstm (LSTM)	(None, 50)	35,800
dropout_1 (Dropout)	(None, 50)	0
dense (Dense)	(None, 20)	1,020
dense_1 (Dense)	(None, 1)	21

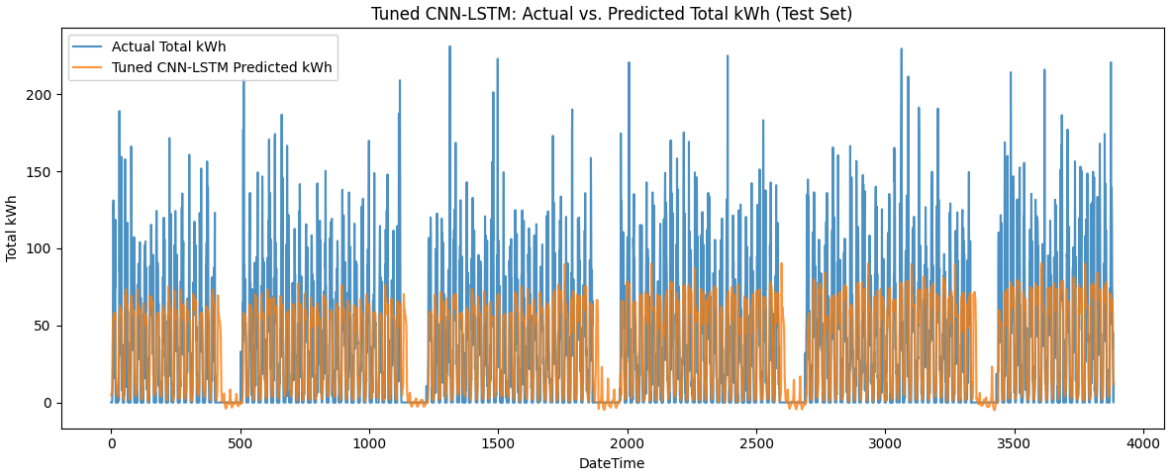
Total params: 42,857 (167.41 KB)

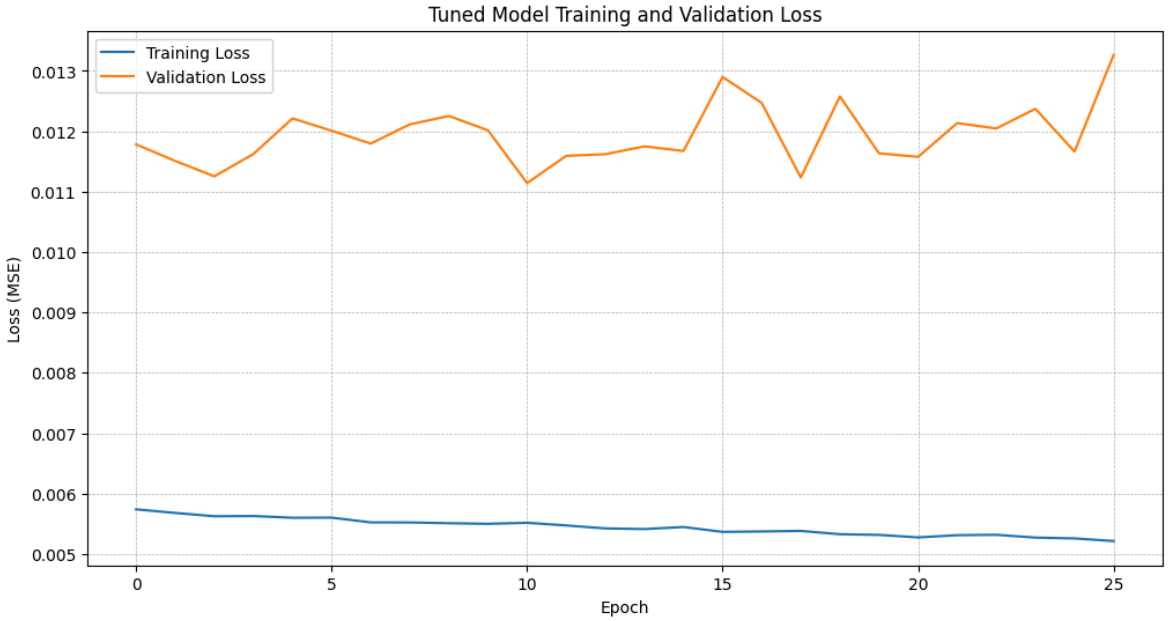
Trainable params: 42,601 (166.41 KB)

Non-trainable params: 256 (1.00 KB)

```
Training the best model for more epochs...
Epoch 1/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 9s 12ms/step - loss: 0.0058 - val_loss: 0.0118
Epoch 2/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0057 - val_loss: 0.0115
Epoch 3/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0056 - val_loss: 0.0113
Epoch 4/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.0055 - val_loss: 0.0116
Epoch 5/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0055 - val_loss: 0.0122
Epoch 6/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0057 - val_loss: 0.0120
Epoch 7/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0055 - val_loss: 0.0118
Epoch 8/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0054 - val_loss: 0.0121
Epoch 9/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0054 - val_loss: 0.0123
Epoch 10/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0055 - val_loss: 0.0120
Epoch 11/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0053 - val_loss: 0.0111
Epoch 12/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 6s 11ms/step - loss: 0.0054 - val_loss: 0.0116
Epoch 13/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0052 - val_loss: 0.0116
Epoch 14/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.0052 - val_loss: 0.0117
Epoch 15/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0056 - val_loss: 0.0117
Epoch 16/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0054 - val_loss: 0.0129
Epoch 17/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0053 - val_loss: 0.0125
Epoch 18/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.0053 - val_loss: 0.0112
Epoch 19/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.0052 - val_loss: 0.0126
Epoch 20/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0051 - val_loss: 0.0116
Epoch 21/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0054 - val_loss: 0.0116
Epoch 22/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0052 - val_loss: 0.0121
Epoch 23/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0052 - val_loss: 0.0120
Epoch 24/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 11ms/step - loss: 0.0052 - val_loss: 0.0124
Epoch 25/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0051 - val_loss: 0.0117
Epoch 26/150
570/570 ━━━━━━━━━━━━━━━━━━━━ 7s 12ms/step - loss: 0.0052 - val_loss: 0.0133
Epoch 26: early stopping
Restoring model weights from the end of the best epoch: 11.

Making predictions on validation and test sets...
122/122 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step
122/122 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Inverse-scaling the data to original units...
```

```
--- Tuned CNN-LSTM Validation Metrics ---
MAE:  16.50
RMSE: 25.78
R^2:  0.574

--- Tuned CNN-LSTM Test Metrics ---
MAE:  18.93
RMSE: 29.44
R^2:  0.547
```





Final Model Comparison

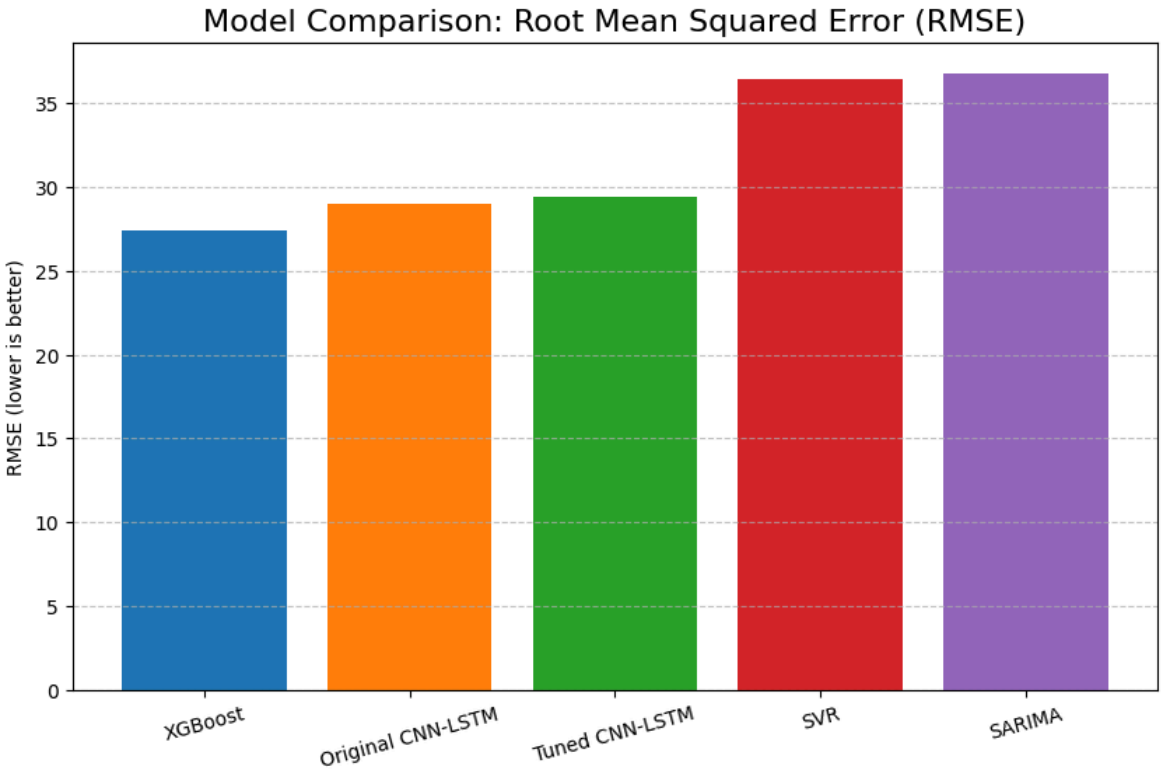
To conclude my analysis, I am now systematically comparing the performance of every model I have trained. I am calculating the final MAE, RMSE, and R-squared metrics for each model on the test set and compiling them into a single summary table. To provide a clear visual summary, I am also generating a bar chart that directly compares the Root Mean Squared Error across all models. This final step allows me to definitively identify the most accurate forecasting model for my dissertation based on quantitative evidence.

In [110…

```
# Final Model Comparison ---
try:
    # Calculate metrics for the Tuned CNN-LSTM model
```

```
    tuned_lstm_mae = mean_absolute_error(y_test_true, y_test_pred)
    tuned_lstm_rmse = np.sqrt(mean_squared_error(y_test_true, y_test_pred))
    tuned_lstm_r2 = r2_score(y_test_true, y_test_pred)

    # Calculate metrics for the Original CNN-LSTM model
    # This now uses 'y_test_pred_original_lstm', which should hold the unscaled
    original_lstm_mae = mean_absolute_error(y_test_true, y_test_pred_original_ls
    original_lstm_rmse = np.sqrt(mean_squared_error(y_test_true, y_test_pred_ori
    original_lstm_r2 = r2_score(y_test_true, y_test_pred_original_lstm)

    # Calculate metrics for other models
    # Note: These models used the original y_test, which has a different length
    # than the y_test_true used for the sequenced LSTM model.
    sarima_mae = mean_absolute_error(y_test, y_test_pred_sarima)
    sarima_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_sarima))
    sarima_r2 = r2_score(y_test, y_test_pred_sarima)

    svr_mae = mean_absolute_error(y_test, y_test_pred_svr)
    svr_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_svr))
    svr_r2 = r2_score(y_test, y_test_pred_svr)

    xgb_mae = mean_absolute_error(y_test, y_test_pred_xgb)
    xgb_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_xgb))
    xgb_r2 = r2_score(y_test, y_test_pred_xgb)

    # Create a DataFrame for comparison
    results_data = {
        'Model': ['SARIMA', 'SVR', 'XGBoost', 'Original CNN-LSTM', 'Tuned CNN-LS
        'MAE': [sarima_mae, svr_mae, xgb_mae, original_lstm_mae, tuned_lstm_mae]
        'RMSE': [sarima_rmse, svr_rmse, xgb_rmse, original_lstm_rmse, tuned_lstm
        'R²': [sarima_r2, svr_r2, xgb_r2, original_lstm_r2, tuned_lstm_r2]
    }
    results_df = pd.DataFrame(results_data).sort_values(by='RMSE').reset_index(d

    print("\n--- Summary of All Model Performances ---")
    print(results_df.to_string())

    # Create a bar plot for visual comparison of RMSE
    plt.figure(figsize=(10, 6))
    plt.bar(results_df['Model'], results_df['RMSE'], color=['#1f77b4', '#ff7f0e'
    plt.title('Model Comparison: Root Mean Squared Error (RMSE)', fontsize=16)
    plt.ylabel('RMSE (lower is better)')
    plt.xticks(rotation=15)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.savefig("RMSE Comparison_visualisation.png", dpi=1200, bbox_inches='tigh
    plt.savefig("Model_comparison_plot.png", dpi=1200, bbox_inches='tight')
    plt.show()

except NameError as e:
    print(f"\nCould not generate final comparison table: {e}")
    print("Please ensure that predictions from all models (e.g., 'y_test_pred_sa
```

```
--- Summary of All Model Performances ---
                Model        MAE       RMSE        R²
0             XGBoost  15.904577  27.431545  0.607949
1  Original CNN-LSTM  19.298081  29.011072  0.560327
2     Tuned CNN-LSTM  18.932786  29.443224  0.547130
3                SVR  21.554392  36.442890  0.308061
4             SARIMA  24.014740  36.789473  0.294838
```



Saving the Final Models

As the final step in my process, I am saving all the machine learning models that I have trained and evaluated. I first create a dedicated directory to store these files neatly. Then, using appropriate methods for each type of model—joblib for scikit-learn, .save for SARIMA, and native functions for XGBoost and TensorFlow/Keras—I save each trained model to a distinct file. This ensures that the fully trained models are preserved and can be reloaded later for future use without needing to repeat the entire training and optimisation process.

In [111…

```
import joblib
import os

# Save All Trained Models ---
print("\n--- Saving All Trained Models with Proper Names ---")

# Create a directory to save the models if it doesn't exist
models_dir = '/content/keras_tuner_dir/ev_demand_forecasting'
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

try:
    # Save scikit-learn and statsmodels models using joblib
    joblib.dump(svr, os.path.join(models_dir, 'ev_demand_svr_model.joblib'))
    print("SVR model saved successfully.")

    sarima_fit.save(os.path.join(models_dir, 'ev_demand_sarima_model.pkl'))
    print("SARIMA model saved successfully.")

    # Save XGBoost model
    best_xgb.save_model(os.path.join(models_dir, 'ev_demand_xgboost_model.json'))
    print("XGBoost model saved successfully.")
```

```
    # Save TensorFlow/Keras models
    model_v2.save(os.path.join(models_dir, 'ev_demand_original_cnnlstm_model.ker
    print("Original CNN-LSTM model saved successfully.")

    best_model.save(os.path.join(models_dir, 'ev_demand_tuned_cnnlstm_model.kera
    print("Tuned CNN-LSTM model saved successfully.")

except NameError as e:
    print(f"\nCould not save all models: {e}")
    print("Please ensure that all model objects (e.g., 'svr', 'best_xgb', 'sarim
```

```
--- Saving All Trained Models with Proper Names ---
SVR model saved successfully.
SARIMA model saved successfully.
XGBoost model saved successfully.
Original CNN-LSTM model saved successfully.
Tuned CNN-LSTM model saved successfully.
```

Loading the Saved Models

I am now creating a helper function to reload all the models I have previously saved. This function will read the model files from the specified directory and load them back into memory, storing them in a dictionary for easy access. This is a crucial step for reproducibility, as it allows me to use the trained models for future analysis or deployment without needing to run the entire training process again. I have also included error handling to manage any issues, for instance, if a model file is missing from the directory.

In [112…

```
# Helper Function to Load All Models ---
# This function loads all the saved models from the 'saved_models' directory.
def load_all_models(models_dir='/content/keras_tuner_dir/ev_demand_forecasting')
    """Loads all saved models from the specified directory."""
    models = {}
    print("--- Loading Saved Models ---")
    try:
        # Load scikit-learn and statsmodels models
        models['SVR'] = joblib.load(os.path.join(models_dir, 'ev_demand_svr_mode
        print("SVR model loaded.")

        # Load XGBoost model
        xgb_model = xgb.XGBRegressor()
        xgb_model.load_model(os.path.join(models_dir, 'ev_demand_xgboost_model.j
        models['XGBoost'] = xgb_model
        print("XGBoost model loaded.")

        # Load TensorFlow/Keras models
        models['Original CNN-LSTM'] = tf.keras.models.load_model(os.path.join(mo
        print("Original CNN-LSTM model loaded.")

        models['Tuned CNN-LSTM'] = tf.keras.models.load_model(os.path.join(model
        print("Tuned CNN-LSTM model loaded.")

    except Exception as e:
        print(f"An error occurred while loading models: {e}")
        print("Please ensure all model files exist in the 'saved_models' directo
        return None
```

```
    return models
```

Creating a Master Visualisation Function

To compare the performance of all my models on a specific day, I am creating a master plotting function. This function takes a single date as input, prepares the necessary data for that day, and then generates predictions from every loaded model—SVR, XGBoost, and both the original and tuned neural networks. It then plots all these predictions on a single graph against the actual energy demand, allowing for a direct, visual comparison of their forecasting accuracy for a 24-hour period. This provides an intuitive and powerful way to assess which model performs best on a given day.

In [113…

```
# Master Plotting Function ---
# This function handles the data preparation and plotting for a specific day.
def plot_day_comparison(target_date_str, models, data_dict, title_prefix=""):
    """
    Generates and plots predictions from all models for a single day.

    Args:
        target_date_str (str): The date to plot, in 'YYYY-MM-DD' format.
        models (dict): A dictionary of the loaded model objects.
        data_dict (dict): A dictionary containing necessary data arrays.
        title_prefix (str): A string to prepend to the plot title.
    """

    # Extract necessary data from the dictionary for easier access
    X_test = data_dict['X_test']
    y_test = data_dict['y_test']
    X_test_scaled = data_dict['X_test_scaled']
    X_test_seq = data_dict['X_test_seq']
    scaler_y = data_dict['scaler_y']
    window_size = X_test_seq.shape[1]

    # --- Data Preparation for the Target Day ---
    target_date = pd.to_datetime(target_date_str)
    day_mask = (X_test.index.date == target_date.date())

    if not day_mask.any():
        print(f"Error: No data found for {target_date_str} in the test set.")
        return

    # Get actual values for the day
    y_actual_day = y_test[day_mask]

    # Get feature data for non-sequential models
    X_test_day = X_test[day_mask]

    # Find the start index for sequential models
    start_idx = np.where(X_test.index == y_actual_day.index[0])[0][0]

    # Create the sequences required for the LSTM models for that day
    X_test_day_seq = []
    for i in range(len(y_actual_day)):
        seq_start_index = start_idx + i - window_size
        if seq_start_index < 0: continue
```

```
        sequence = X_test_scaled[seq_start_index : start_idx + i]
        X_test_day_seq.append(sequence)

    if not X_test_day_seq:
        print(f"Could not generate sequences for {target_date_str}. Not enough p
        return

    X_test_day_seq = np.array(X_test_day_seq)

    # --- Generate Predictions ---
    predictions = {}
    for name, model in models.items():
        print(f"Generating predictions for {name}...")
        if "CNN-LSTM" in name:
            pred_scaled = model.predict(X_test_day_seq, verbose=0)
            predictions[name] = scaler_y.inverse_transform(pred_scaled).flatten(
        else:
            predictions[name] = model.predict(X_test_day)

    # --- Plotting ---
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(14, 7))

    plt.plot(y_actual_day.values, 'o-', label='Actual Demand', color='black', li

    colors = {'SVR': 'green', 'XGBoost': 'blue', 'Original CNN-LSTM': 'purple',
    for name, pred_values in predictions.items():
        plt.plot(pred_values, 's--', label=f'{name} Prediction', color=colors.ge

    plt.title(f'{title_prefix}: {target_date.strftime("%A, %Y-%m-%d")}', fontsiz
    plt.xlabel('Hour of the Day', fontsize=14)
    plt.ylabel('Total kWh', fontsize=14)
    plt.xticks(ticks=np.arange(0, 24, 2))
    plt.legend(fontsize=12)
    plt.ylim(bottom=0)
    plt.savefig("Saved prediction_visualisation.png", dpi=1200, bbox_inches='tig
    plt.show()
```

Preparing for Final Visualisation

I am now setting the stage for the final visual comparison of my models. I first call the function I just created to load all the saved models back into the programming environment. I then group all the necessary data arrays—such as the various test sets and the data scaler—into a single dictionary. This organises the data neatly, making it easy to pass all the required information to the master plotting function in the next step.

In [114…

```
# Load Models and Prepare Data
# This block must be run first. It assumes all the necessary data variables
# from your notebook (X_test, y_test, etc.) are available.
try:
    loaded_models = load_all_models()

    data_for_plotting = {
        'X_test': X_test,
        'y_test': y_test,
        'X_test_scaled': X_test_scaled,
```

```
        'X_test_seq': X_test_seq,
        'scaler_y': scaler_y
    }
except NameError as e:
    print(f"\nExecution failed: {e}")
    print("Please ensure you have run all previous cells in your notebook to def
    loaded_models = None
```

```
--- Loading Saved Models ---
SVR model loaded.
XGBoost model loaded.
Original CNN-LSTM model loaded.
Tuned CNN-LSTM model loaded.
```

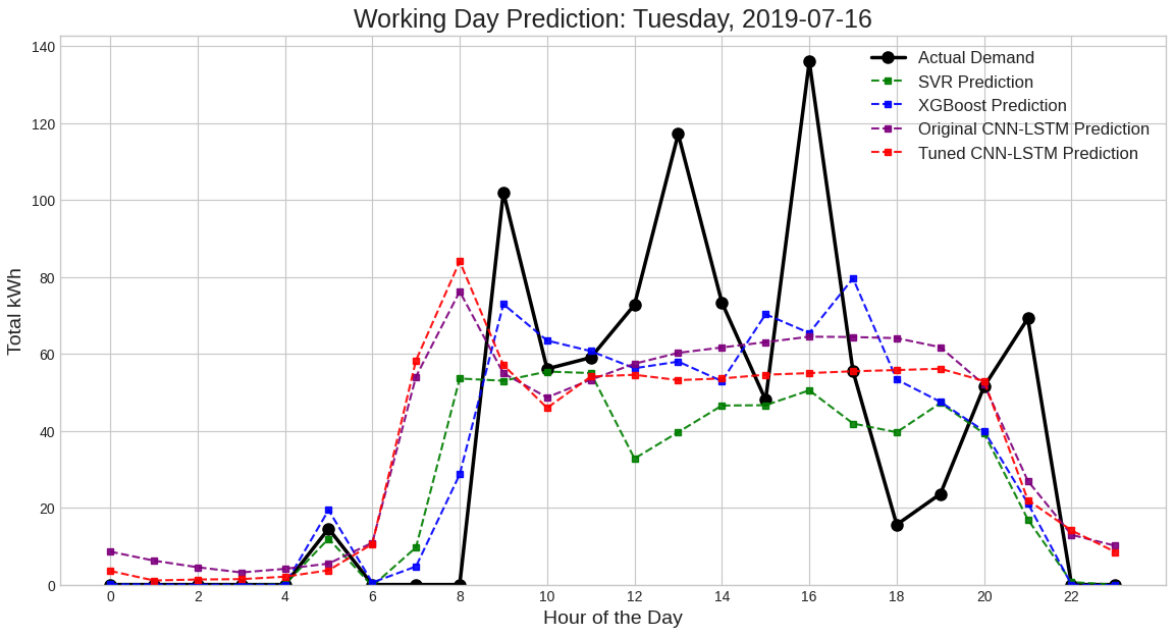
Generating Final Comparison Plots

I am now using the master plotting function to generate a series of visualisations that directly compare the performance of all my trained models. I am deliberately selecting three distinct and representative days from the test set: a typical working day, a weekend day, and a bank holiday. This allows me to assess how each model performs under different real-world demand patterns and provides a final, qualitative evaluation of their forecasting capabilities in a range of scenarios.

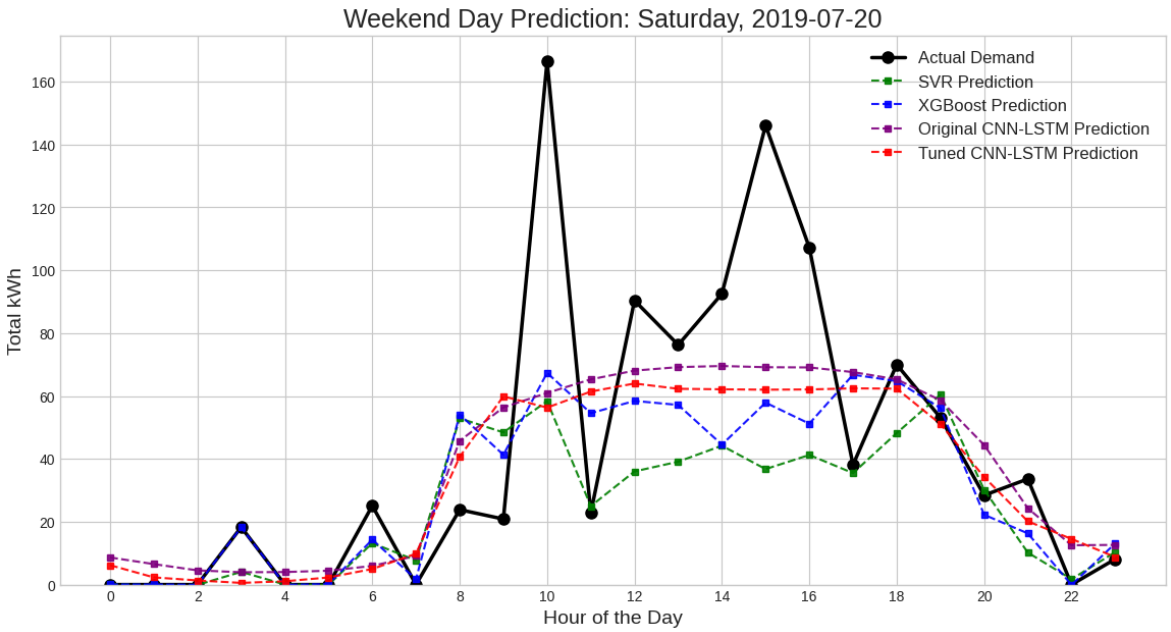
In [115…

```
# Generate All Plots ---
if loaded_models:
    # Plot for a Working Day
    plot_day_comparison(
        target_date_str='2019-07-16',
        models=loaded_models,
        data_dict=data_for_plotting,
        title_prefix="Working Day Prediction"
    )

    # Plot for a Weekend Day
    plot_day_comparison(
        target_date_str='2019-07-20',
        models=loaded_models,
        data_dict=data_for_plotting,
        title_prefix="Weekend Day Prediction"
    )

    # Plot for a Bank Holiday
    plot_day_comparison(
        target_date_str='2019-08-26',
        models=loaded_models,
        data_dict=data_for_plotting,
        title_prefix="Bank Holiday Prediction"
    )
```

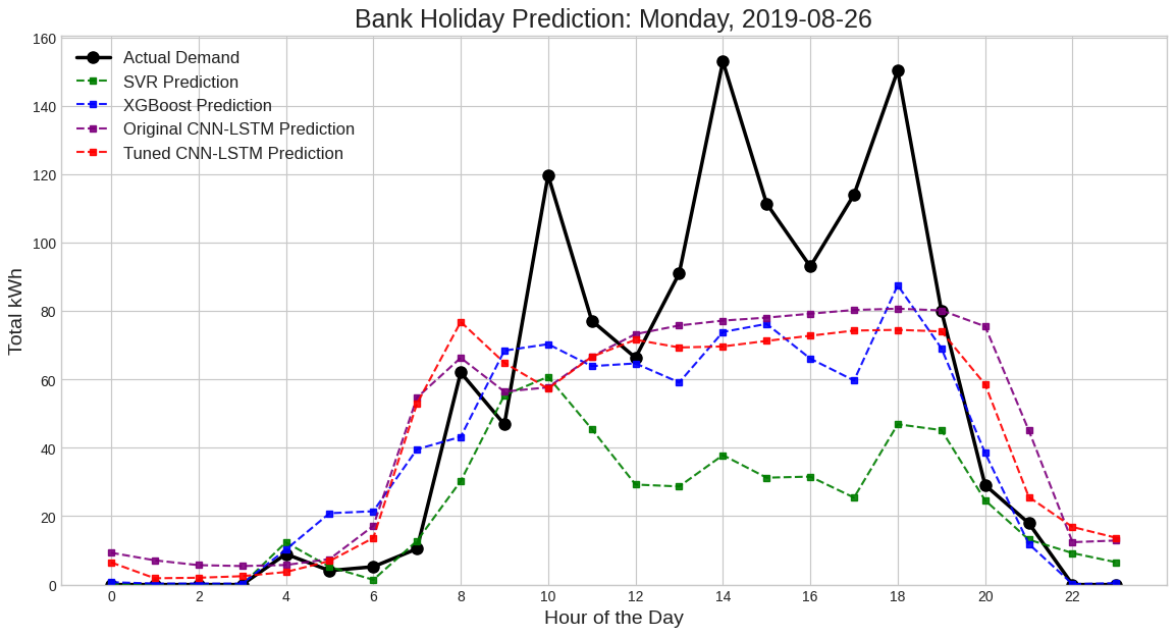
```
Generating predictions for SVR...
Generating predictions for XGBoost...
Generating predictions for Original CNN-LSTM...
Generating predictions for Tuned CNN-LSTM...
```




```
Generating predictions for SVR...
Generating predictions for XGBoost...
Generating predictions for Original CNN-LSTM...
Generating predictions for Tuned CNN-LSTM...
```




```
Generating predictions for SVR...
Generating predictions for XGBoost...
Generating predictions for Original CNN-LSTM...
Generating predictions for Tuned CNN-LSTM...
```



In [116…

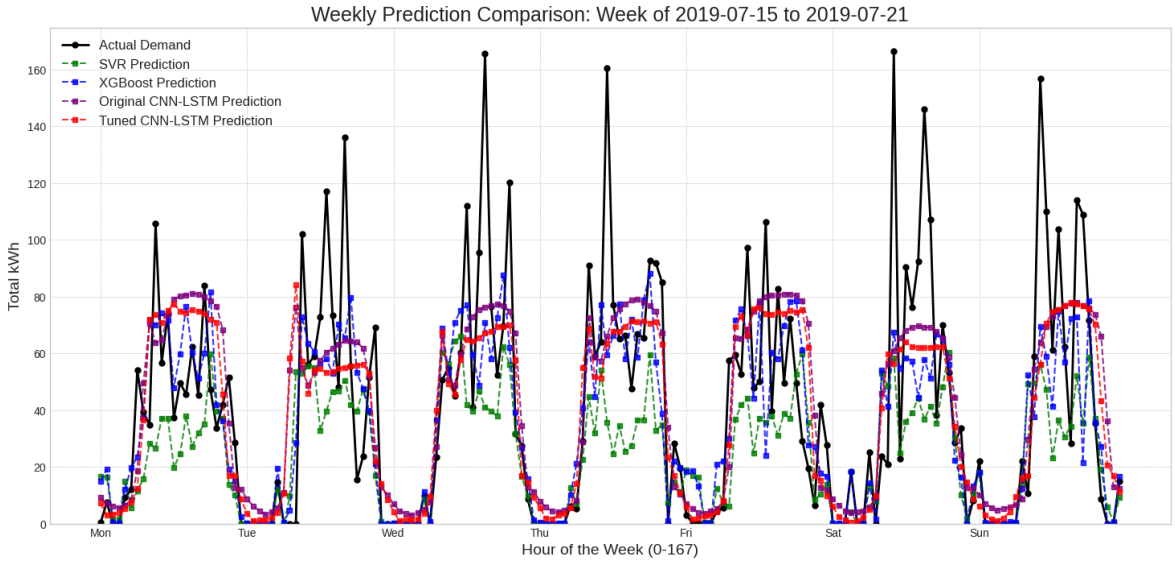
```
# This function handles the data preparation and plotting for a specific 7-day p
def plot_week_comparison(start_date_str, models, data_dict, title_prefix=""):
    """
    Generates and plots predictions from all models for a single week.

    Args:
        start_date_str (str): The start date of the week to plot, in 'YYYY-MM-DD
        models (dict): A dictionary of the loaded model objects.
        data_dict (dict): A dictionary containing necessary data arrays.
        title_prefix (str): A string to prepend to the plot title.
    """

    # Extract necessary data from the dictionary
    X_test = data_dict['X_test']
    y_test = data_dict['y_test']
    X_test_scaled = data_dict['X_test_scaled']
    X_test_seq = data_dict['X_test_seq']
    scaler_y = data_dict['scaler_y']
    window_size = X_test_seq.shape[1]

    # --- Data Preparation for the Target Week ---
    start_date = pd.to_datetime(start_date_str)
    end_date = start_date + pd.Timedelta(days=6)
    week_mask = (X_test.index.date >= start_date.date()) & (X_test.index.date <=

    if not week_mask.any():
        print(f"Error: No data found for the week starting {start_date_str} in t
        return

    # Get actual values and feature data for the week
    y_actual_week = y_test[week_mask]
    X_test_week = X_test[week_mask]

    # Find the start index for sequential models
    try:
        start_idx = np.where(X_test.index == y_actual_week.index[0])[0][0]
    except IndexError:
        print(f"Error: Could not find the start of the week {start_date_str} in
        return
```

```
    # Create the sequences required for the LSTM models for the entire week
    X_test_week_seq = []
    for i in range(len(y_actual_week)):
        seq_start_index = start_idx + i - window_size
        if seq_start_index < 0: continue

        sequence = X_test_scaled[seq_start_index : start_idx + i]
        X_test_week_seq.append(sequence)

    if not X_test_week_seq:
        print(f"Could not generate sequences for the week starting {start_date_s
        return

    X_test_week_seq = np.array(X_test_week_seq)

    # --- Generate Predictions ---
    predictions = {}
    for name, model in models.items():
        print(f"Generating predictions for {name}...")
        if "CNN-LSTM" in name:
            pred_scaled = model.predict(X_test_week_seq, verbose=0)
            predictions[name] = scaler_y.inverse_transform(pred_scaled).flatten(
        else:
            predictions[name] = model.predict(X_test_week)

    # --- Plotting ---
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(18, 8))

    plt.plot(y_actual_week.values, 'o-', label='Actual Demand', color='black', l

    colors = {'SVR': 'green', 'XGBoost': 'blue', 'Original CNN-LSTM': 'purple',
    for name, pred_values in predictions.items():
        plt.plot(pred_values, 's--', label=f'{name} Prediction', color=colors.ge

    plt.title(f'{title_prefix}: Week of {start_date.strftime("%Y-%m-%d")} to {en
    plt.xlabel('Hour of the Week (0-167)', fontsize=14)
    plt.ylabel('Total kWh', fontsize=14)
    plt.xticks(ticks=np.arange(0, 168, 24), labels=['Mon', 'Tue', 'Wed', 'Thu',
    plt.legend(fontsize=12)
    plt.ylim(bottom=0)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.savefig("Saved visuals_visualisation.png", dpi=1200, bbox_inches='tight'
    plt.show()
```

In [117…

```
if loaded_models:
    # Plot for a sample week from the test data
    plot_week_comparison(
        start_date_str='2019-07-15', # A sample week in July
        models=loaded_models,
        data_dict=data_for_plotting,
        title_prefix="Weekly Prediction Comparison"
    )
```

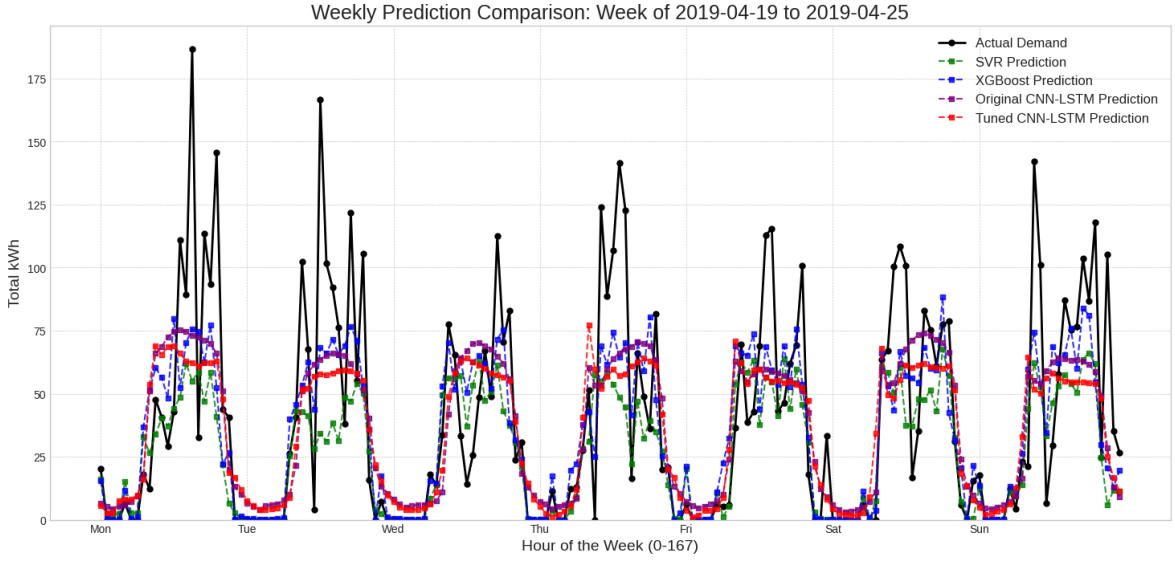
```
Generating predictions for SVR...
Generating predictions for XGBoost...
Generating predictions for Original CNN-LSTM...
Generating predictions for Tuned CNN-LSTM...
```



In [118…

```
if loaded_models:
    # Plot for a sample week from the test data
    plot_week_comparison(
        start_date_str='2019-04-19', # A sample week in July
        models=loaded_models,
        data_dict=data_for_plotting,
        title_prefix="Weekly Prediction Comparison"
    )
```

```
Generating predictions for SVR...
Generating predictions for XGBoost...
Generating predictions for Original CNN-LSTM...
Generating predictions for Tuned CNN-LSTM...
```



In [ ]:

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.4.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.16.4-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from optuna) (25.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.11/dis
t-packages (from optuna) (2.0.42)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (f
rom optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages
(from optuna) (6.0.2)
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from alemb
ic>=1.5.0->optuna) (1.1.3)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.
11/dist-packages (from alembic>=1.5.0->optuna) (4.14.1)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-pack
ages (from sqlalchemy>=1.4.2->optuna) (3.2.3)
Downloading optuna-4.4.0-py3-none-any.whl (395 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 395.9/395.9 kB 19.9 MB/s eta 0:00:00
Downloading alembic-1.16.4-py3-none-any.whl (247 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.0/247.0 kB 26.9 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, alembic, optuna
Successfully installed alembic-1.16.4 colorlog-6.9.0 optuna-4.4.0
```

Optimising the XGBoost Model with Optuna

To find the best possible configuration for my XGBoost model, I am using an advanced optimisation framework called Optuna. I first define an 'objective' function that trains an XGBoost model using a specific set of hyperparameters suggested by Optuna and then returns the model's error on the validation set.

Optuna then automatically runs 50 trials, intelligently choosing new hyperparameter combinations each time to try and minimise this error. Once the search is complete and the best parameters are identified, I train a final model on the combined training and validation data. I then evaluate this fully optimised model on the test set to get a definitive measure of its performance and save the final model to a file for future use.

In [120…

```
import optuna
# Define the Objective Function for Optuna ---
def objective(trial):
    params = {
        'objective': 'reg:squarederror', 'eval_metric': 'rmse',
        'n_estimators': trial.suggest_int('n_estimators', 400, 1500),
        'learning_rate': trial.suggest_float('learning_rate', 1e-3, 0.1, log=Tru
        'max_depth': trial.suggest_int('max_depth', 3, 10),
        'subsample': trial.suggest_float('subsample', 0.6, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 10),
        'gamma': trial.suggest_float('gamma', 1e-8, 1.0, log=True),
        'lambda': trial.suggest_float('lambda', 1e-8, 1.0, log=True),
```

```
        'alpha': trial.suggest_float('alpha', 1e-8, 1.0, log=True),
        'seed': 42
    }
    model = xgb.XGBRegressor(**params)
    model.fit(X_train, y_train, eval_set=[(X_val, y_val)], verbose=False)
    preds = model.predict(X_val)
    rmse = np.sqrt(mean_squared_error(y_val, preds))
    return rmse

# Create and Run the Optuna Study ---
print("--- Starting XGBoost Hyperparameter Optimization with Optuna ---")
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)

# Display the Best Results ---
print("\n--- Optimization Complete ---")
print(f"Number of finished trials: {len(study.trials)}")
print("Best trial:")
best_trial = study.best_trial
print(f"  Value (Validation RMSE): {best_trial.value:.4f}")
print("  Params: ")
for key, value in best_trial.params.items():
    print(f"    {key}: {value}")

# Train Final Model with Best Parameters and Evaluate ---
print("\n--- Training Final Model with Optimal Parameters ---")
best_params = best_trial.params
final_xgb_model = xgb.XGBRegressor(**best_params, seed=42)
X_train_val = pd.concat([X_train, X_val])
y_train_val = pd.concat([y_train, y_val])
final_xgb_model.fit(X_train_val, y_train_val, verbose=False)

# Evaluate on Validation and Test Sets ---
y_val_pred_optimized = final_xgb_model.predict(X_val)
y_test_pred_optimized = final_xgb_model.predict(X_test)

val_mae = mean_absolute_error(y_val, y_val_pred_optimized)
val_rmse = np.sqrt(mean_squared_error(y_val, y_val_pred_optimized))
val_r2 = r2_score(y_val, y_val_pred_optimized)

test_mae = mean_absolute_error(y_test, y_test_pred_optimized)
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred_optimized))
test_r2 = r2_score(y_test, y_test_pred_optimized)

print("\n--- Final Optimized Model Performance on Validation Set ---")
print(f"MAE: {val_mae:.4f}")
print(f"RMSE: {val_rmse:.4f}")
print(f"R^2:  {val_r2:.4f}")

print("\n--- Final Optimized Model Performance on Test Set ---")
print(f"MAE: {test_mae:.4f}")
print(f"RMSE: {test_rmse:.4f}")
print(f"R^2:  {test_r2:.4f}")

# Save the Optimized Model ---
print("\n--- Saving the Optimized XGBoost Model ---")
models_dir = '/content/keras_tuner_dir/ev_demand_forecasting'
if not os.path.exists(models_dir):
    os.makedirs(models_dir)
```

```
final_xgb_model.save_model(os.path.join(models_dir, 'ev_demand_optimized_xgboost
print(f"Optimized model saved to '{os.path.join(models_dir, 'ev_demand_optimized
```

```
[I 2025-08-16 17:18:02,737] A new study created in memory with name: no-name-3ff5
0856-828e-4669-b76b-bef3eb744ed8
--- Starting XGBoost Hyperparameter Optimization with Optuna ---
```

```
[I 2025-08-16 17:18:06,024] Trial 0 finished with value: 23.222681345769494 and p
arameters: {'n_estimators': 831, 'learning_rate': 0.009718959918100418, 'max_dept
h': 7, 'subsample': 0.7389441018183428, 'colsample_bytree': 0.7022469225072678,
'min_child_weight': 7, 'gamma': 0.897167863764948, 'lambda': 0.001714032671442268
9, 'alpha': 0.0214555676507322l}. Best is trial 0 with value: 23.222681345769494.
[I 2025-08-16 17:18:18,065] Trial 1 finished with value: 24.246059718777357 and p
arameters: {'n_estimators': 1146, 'learning_rate': 0.01622907616805015, 'max_dept
h': 8, 'subsample': 0.9199398702525808, 'colsample_bytree': 0.8510706314004777,
'min_child_weight': 3, 'gamma': 1.5664466035184565e-06, 'lambda': 9.5787386283158
94e-07, 'alpha': 0.0011308109932120366}. Best is trial 0 with value: 23.222681345
769494.
[I 2025-08-16 17:18:39,097] Trial 2 finished with value: 25.65089439385825 and pa
rameters: {'n_estimators': 1334, 'learning_rate': 0.07541162179572557, 'max_dept
h': 5, 'subsample': 0.8018289326830396, 'colsample_bytree': 0.7689699790573052,
'min_child_weight': 9, 'gamma': 4.006071203573545e-05, 'lambda': 0.00139620892918
84386, 'alpha': 0.13506056106554118}. Best is trial 0 with value: 23.222681345769
494.
[I 2025-08-16 17:19:04,255] Trial 3 finished with value: 24.93578866004467 and pa
rameters: {'n_estimators': 1362, 'learning_rate': 0.0015916173836695053, 'max_dep
th': 5, 'subsample': 0.9705717624291729, 'colsample_bytree': 0.9929285380993074,
'min_child_weight': 8, 'gamma': 6.451401204638452e-06, 'lambda': 5.89933053251045
4e-07, 'alpha': 0.00045608371718230473}. Best is trial 0 with value: 23.222681345
769494.
[I 2025-08-16 17:19:06,266] Trial 4 finished with value: 23.409250119242543 and p
arameters: {'n_estimators': 902, 'learning_rate': 0.008934795881262725, 'max_dept
h': 3, 'subsample': 0.884891555880878, 'colsample_bytree': 0.8560524493192809, 'm
in_child_weight': 6, 'gamma': 8.372199812820016e-07, 'lambda': 0.0001064768791690
9827, 'alpha': 1.3221977233941364e-05}. Best is trial 0 with value: 23.2226813457
69494.
[I 2025-08-16 17:19:14,471] Trial 5 finished with value: 24.83313545914557 and pa
rameters: {'n_estimators': 859, 'learning_rate': 0.025166686521270822, 'max_dept
h': 10, 'subsample': 0.8935767104050851, 'colsample_bytree': 0.9954660052447998,
'min_child_weight': 10, 'gamma': 1.3485903870328644e-07, 'lambda': 0.813458032755
1907, 'alpha': 0.1181017294658585}. Best is trial 0 with value: 23.22268134576949
4.
[I 2025-08-16 17:19:20,178] Trial 6 finished with value: 23.503324500663265 and p
arameters: {'n_estimators': 1496, 'learning_rate': 0.01004506164244078, 'max_dept
h': 7, 'subsample': 0.854572915802646, 'colsample_bytree': 0.6180577665000819, 'm
in_child_weight': 10, 'gamma': 0.2889551599476855, 'lambda': 1.2293565769617495e-
05, 'alpha': 3.444608202891357e-05}. Best is trial 0 with value: 23.2226813457694
94.
[I 2025-08-16 17:19:27,723] Trial 7 finished with value: 23.283282878797245 and p
arameters: {'n_estimators': 1500, 'learning_rate': 0.008198686898901623, 'max_dep
th': 5, 'subsample': 0.8882757658161824, 'colsample_bytree': 0.6932160577485347,
'min_child_weight': 5, 'gamma': 2.909569737222314e-05, 'lambda': 0.00883249712425
98, 'alpha': 0.3825062097930709}. Best is trial 0 with value: 23.222681345769494.
[I 2025-08-16 17:19:30,340] Trial 8 finished with value: 23.687031777045735 and p
arameters: {'n_estimators': 540, 'learning_rate': 0.004531994129808686, 'max_dept
h': 8, 'subsample': 0.833895329665201, 'colsample_bytree': 0.7684447120990039, 'm
in_child_weight': 10, 'gamma': 0.003974033966217523, 'lambda': 0.0413867271215846
4, 'alpha': 9.241669335135511e-05}. Best is trial 0 with value: 23.22268134576949
4.
[I 2025-08-16 17:19:34,553] Trial 9 finished with value: 22.97052225950021 and pa
rameters: {'n_estimators': 1347, 'learning_rate': 0.0036400392821662165, 'max_dep
th': 5, 'subsample': 0.8987871547609226, 'colsample_bytree': 0.6696523050447284,
'min_child_weight': 5, 'gamma': 0.20573812502919875, 'lambda': 0.0009336156804846
567, 'alpha': 0.0006792276075641363}. Best is trial 9 with value: 22.970522259500
21.
[I 2025-08-16 17:19:40,660] Trial 10 finished with value: 30.59871706327347 and p
arameters: {'n_estimators': 1156, 'learning_rate': 0.0010432956070712777, 'max_de
```

```
pth': 3, 'subsample': 0.601723586721799, 'colsample_bytree': 0.609135668825207,
'min_child_weight': 2, 'gamma': 0.004011049203143244, 'lambda': 1.583923776942073
e-08, 'alpha': 3.102435241641891e-08}. Best is trial 9 with value: 22.97052225950
021.
[I 2025-08-16 17:19:43,046] Trial 11 finished with value: 25.243660007742786 and
parameters: {'n_estimators': 667, 'learning_rate': 0.0028744507878270685, 'max_de
pth': 6, 'subsample': 0.7106126535135345, 'colsample_bytree': 0.6983068242727027,
'min_child_weight': 6, 'gamma': 0.7504660777148671, 'lambda': 0.0003350571599736783,
'alpha': 0.009086612943052292}. Best is trial 9 with value: 22.970522259500
1.
[I 2025-08-16 17:19:46,254] Trial 12 finished with value: 23.91506867120978 and p
arameters: {'n_estimators': 770, 'learning_rate': 0.0032642227116462972, 'max_dep
th': 7, 'subsample': 0.720360063093223, 'colsample_bytree': 0.6878540284621576,
'min_child_weight': 4, 'gamma': 0.044627954317086896, 'lambda': 0.019498251518808
213, 'alpha': 0.0069733473308608955}. Best is trial 9 with value: 22.970522259500
21.
[I 2025-08-16 17:19:55,970] Trial 13 finished with value: 24.995133133647112 and
parameters: {'n_estimators': 1063, 'learning_rate': 0.03166569112775909, 'max_dep
th': 10, 'subsample': 0.7322633214992585, 'colsample_bytree': 0.67400648122078,
'min_child_weight': 7, 'gamma': 0.0023734614732048077, 'lambda': 1.4004560334576497e-05, 'alpha': 2.391132473757301e-06}. Best is trial 9 with value: 22.970522259
50021.
[I 2025-08-16 17:19:57,192] Trial 14 finished with value: 24.97115446813362 and p
arameters: {'n_estimators': 463, 'learning_rate': 0.005489649553525541, 'max_dept
h': 4, 'subsample': 0.644065614769751, 'colsample_bytree': 0.7659915127247959, 'm
in_child_weight': 1, 'gamma': 0.04927648931184659, 'lambda': 0.35282932557236646,
'alpha': 0.008488057070765786}. Best is trial 9 with value: 22.97052225950021.
[I 2025-08-16 17:20:02,315] Trial 15 finished with value: 25.03449981441408 and p
arameters: {'n_estimators': 1000, 'learning_rate': 0.0019468014028290786, 'max_de
pth': 8, 'subsample': 0.9916111728095045, 'colsample_bytree': 0.6462127105707797,
'min_child_weight': 4, 'gamma': 0.000336230095697069, 'lambda': 0.001143857555226
685, 'alpha': 1.8328743911379169e-06}. Best is trial 9 with value: 22.97052225950
021.
[I 2025-08-16 17:20:04,696] Trial 16 finished with value: 23.19375681683782 and p
arameters: {'n_estimators': 705, 'learning_rate': 0.015271114427889181, 'max_dept
h': 6, 'subsample': 0.7678290891142302, 'colsample_bytree': 0.7319754390127534,
'min_child_weight': 8, 'gamma': 0.08557095311910339, 'lambda': 0.0035267938373062
385, 'alpha': 0.0008299866273813823}. Best is trial 9 with value: 22.970522259500
21.
[I 2025-08-16 17:20:07,033] Trial 17 finished with value: 24.724399973174673 and
parameters: {'n_estimators': 660, 'learning_rate': 0.05653998187610048, 'max_dept
h': 6, 'subsample': 0.7829619270249663, 'colsample_bytree': 0.8590700796488638,
'min_child_weight': 8, 'gamma': 0.052812889823174086, 'lambda': 2.114977149040050
5e-05, 'alpha': 0.0007276959565684942}. Best is trial 9 with value: 22.9705222595
0021.
[I 2025-08-16 17:20:13,336] Trial 18 finished with value: 24.260779009252015 and
parameters: {'n_estimators': 1278, 'learning_rate': 0.024329908424806406, 'max_de
pth': 4, 'subsample': 0.948881962324316, 'colsample_bytree': 0.7355937877040358,
'min_child_weight': 5, 'gamma': 1.9621414676603162e-08, 'lambda': 0.1014244193821
9234, 'alpha': 7.747194301454811e-08}. Best is trial 9 with value: 22.97052225950
021.
[I 2025-08-16 17:20:15,022] Trial 19 finished with value: 23.132549477956477 and
parameters: {'n_estimators': 674, 'learning_rate': 0.014412878161291087, 'max_dep
th': 4, 'subsample': 0.7785769062338562, 'colsample_bytree': 0.8160051095413026,
'min_child_weight': 8, 'gamma': 0.00038408783336025566, 'lambda': 0.0075777845394
058505, 'alpha': 4.776934765912089e-06}. Best is trial 9 with value: 22.970522259
50021.
[I 2025-08-16 17:20:16,468] Trial 20 finished with value: 24.701416267510194 and
parameters: {'n_estimators': 557, 'learning_rate': 0.004663791139478642, 'max_dep
th': 4, 'subsample': 0.8274200116605759, 'colsample_bytree': 0.9274974349147201,
```

```
'min_child_weight': 7, 'gamma': 0.000328163807641896, 'lambda': 0.000178178596794
85076, 'alpha': 3.851997844858021e-07}. Best is trial 9 with value: 22.9705222595
0021.
[I 2025-08-16 17:20:18,651] Trial 21 finished with value: 23.164616326178457 and
parameters: {'n_estimators': 721, 'learning_rate': 0.015432371248638638, 'max_dep
th': 5, 'subsample': 0.7748795668655089, 'colsample_bytree': 0.823608145279228,
'min_child_weight': 8, 'gamma': 0.016052452271483053, 'lambda': 0.007014649660341
369, 'alpha': 4.636981211738518e-06}. Best is trial 9 with value: 22.970522259500
21.
[I 2025-08-16 17:20:20,320] Trial 22 finished with value: 23.416739330162716 and
parameters: {'n_estimators': 585, 'learning_rate': 0.03695678911021023, 'max_dept
h': 4, 'subsample': 0.6781437706243738, 'colsample_bytree': 0.8137668235685708,
'min_child_weight': 9, 'gamma': 0.00866061117657219, 'lambda': 0.069247394954088
9, 'alpha': 5.13524035135788e-06}. Best is trial 9 with value: 22.97052225950021.
[I 2025-08-16 17:20:21,550] Trial 23 finished with value: 22.900311960932758 and
parameters: {'n_estimators': 407, 'learning_rate': 0.01710868295632237, 'max_dept
h': 5, 'subsample': 0.8031165743373802, 'colsample_bytree': 0.81852546281357, 'mi
n_child_weight': 6, 'gamma': 0.0005053765715811736, 'lambda': 0.00838214937855388
6, 'alpha': 2.0257918762720534e-07}. Best is trial 23 with value: 22.900311960932
758.
[I 2025-08-16 17:20:23,337] Trial 24 finished with value: 25.482100013780805 and
parameters: {'n_estimators': 415, 'learning_rate': 0.0065325731538466505, 'max_de
pth': 3, 'subsample': 0.8522897665386648, 'colsample_bytree': 0.9128605031878636,
'min_child_weight': 4, 'gamma': 0.00043897414275665054, 'lambda': 0.0003302144539
521591, 'alpha': 2.5351704570293045e-07}. Best is trial 23 with value: 22.9003119
60932758.
[I 2025-08-16 17:20:26,771] Trial 25 finished with value: 22.865566638396167 and
parameters: {'n_estimators': 483, 'learning_rate': 0.013566304064134492, 'max_dep
th': 5, 'subsample': 0.8116632658985677, 'colsample_bytree': 0.8920946532919332,
'min_child_weight': 6, 'gamma': 0.000992673913427959, 'lambda': 0.200347207359741
49, 'alpha': 1.2159649353171635e-08}. Best is trial 25 with value: 22.86556663839
6167.
[I 2025-08-16 17:20:27,990] Trial 26 finished with value: 27.63472496781129 and p
arameters: {'n_estimators': 404, 'learning_rate': 0.0031708660444854406, 'max_dep
th': 5, 'subsample': 0.932890629304608, 'colsample_bytree': 0.9069714111534506,
'min_child_weight': 6, 'gamma': 0.00097931919882359 69, 'lambda': 0.44860649707354
333, 'alpha': 1.8235471466553627e-08}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:20:29,738] Trial 27 finished with value: 24.369398691717667 and
parameters: {'n_estimators': 506, 'learning_rate': 0.047433189071797215, 'max_dep
th': 6, 'subsample': 0.8117382520479398, 'colsample_bytree': 0.8725205596763599,
'min_child_weight': 5, 'gamma': 9.427152712594919e-05, 'lambda': 0.08906719020007
106, 'alpha': 2.769073933389477e-07}. Best is trial 25 with value: 22.86556663839
6167.
[I 2025-08-16 17:20:32,865] Trial 28 finished with value: 24.672890893160826 and
parameters: {'n_estimators': 1144, 'learning_rate': 0.02190097063821394, 'max_dep
th': 5, 'subsample': 0.8633650386698192, 'colsample_bytree': 0.9448280603030206,
'min_child_weight': 3, 'gamma': 1.096454109891854e-05, 'lambda': 0.19340752359101
34, 'alpha': 6.716417515303954e-08}. Best is trial 25 with value: 22.865566638396
167.
[I 2025-08-16 17:20:36,142] Trial 29 finished with value: 23.399631403383488 and
parameters: {'n_estimators': 971, 'learning_rate': 0.011857112312066893, 'max_dep
th': 6, 'subsample': 0.7474262693817363, 'colsample_bytree': 0.8791319352584959,
'min_child_weight': 6, 'gamma': 0.2946862506569421, 'lambda': 0.0227589421302228
9, 'alpha': 1.0186883331268267e-08}. Best is trial 25 with value: 22.865566638396
167.
[I 2025-08-16 17:20:41,315] Trial 30 finished with value: 25.84514048439492 and p
arameters: {'n_estimators': 608, 'learning_rate': 0.09949962483065565, 'max_dept
h': 7, 'subsample': 0.9120048427128258, 'colsample_bytree': 0.9586580078395871,
'min_child_weight': 7, 'gamma': 0.017646881733907042, 'lambda': 0.000934786727176
```

```
7802, 'alpha': 0.00011321923427730556}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:20:42,700] Trial 31 finished with value: 23.10761974408953 and p
arameters: {'n_estimators': 487, 'learning_rate': 0.012963009565174705, 'max_dept
h': 4, 'subsample': 0.7971993508044326, 'colsample_bytree': 0.7928818568568046,
'min_child_weight': 5, 'gamma': 0.0010387391909461557, 'lambda': 0.00380844454204
5042, 'alpha': 6.988428074686753e-07}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:20:43,947] Trial 32 finished with value: 23.121978338305635 and
parameters: {'n_estimators': 484, 'learning_rate': 0.019517427641950462, 'max_dep
th': 4, 'subsample': 0.8203483393303073, 'colsample_bytree': 0.7985360441193599,
'min_child_weight': 5, 'gamma': 0.0010206112093624315, 'lambda': 0.00234624322564
2026, 'alpha': 7.336247023299514e-07}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:20:45,232] Trial 33 finished with value: 23.007105338571737 and
parameters: {'n_estimators': 401, 'learning_rate': 0.010832601444788578, 'max_dep
th': 5, 'subsample': 0.7963741371634789, 'colsample_bytree': 0.7396822214026103,
'min_child_weight': 3, 'gamma': 4.4419865063119434e-05, 'lambda': 0.0195944753995
0702, 'alpha': 1.0365714469181205e-07}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:20:46,462] Trial 34 finished with value: 23.935124826672 and par
ameters: {'n_estimators': 415, 'learning_rate': 0.006892370446561899, 'max_dept
h': 5, 'subsample': 0.7567618718544157, 'colsample_bytree': 0.7296649144789189,
'min_child_weight': 3, 'gamma': 6.816760974579544e-05, 'lambda': 0.02292371769273
9792, 'alpha': 7.788768790306463e-08}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:20:50,121] Trial 35 finished with value: 23.196777611071614 and
parameters: {'n_estimators': 1237, 'learning_rate': 0.007700458794910815, 'max_de
pth': 5, 'subsample': 0.6990498771306041, 'colsample_bytree': 0.6417584635746321,
'min_child_weight': 2, 'gamma': 3.265399300250048e-06, 'lambda': 0.00066565968726
78115, 'alpha': 0.0002238216344576635}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:20:57,583] Trial 36 finished with value: 24.287078971017504 and
parameters: {'n_estimators': 1392, 'learning_rate': 0.018246658053656026, 'max_de
pth': 6, 'subsample': 0.8672543722643062, 'colsample_bytree': 0.7526465238342864,
'min_child_weight': 4, 'gamma': 1.40690803540153e-05, 'lambda': 6.832975216526191
e-05, 'alpha': 3.331621814688888e-08}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:20:59,859] Trial 37 finished with value: 23.12471624011385 and p
arameters: {'n_estimators': 807, 'learning_rate': 0.01070378978777766, 'max_dept
h': 5, 'subsample': 0.8036515199313049, 'colsample_bytree': 0.8343641900565083,
'min_child_weight': 2, 'gamma': 2.272540187521822e-07, 'lambda': 0.19098816400836
668, 'alpha': 0.0016313563087798058}. Best is trial 25 with value: 22.86556663839
6167.
[I 2025-08-16 17:21:01,168] Trial 38 finished with value: 29.36835786729345 and p
arameters: {'n_estimators': 598, 'learning_rate': 0.0022549936931910482, 'max_dep
th': 3, 'subsample': 0.8364448748499859, 'colsample_bytree': 0.7195358086339321,
'min_child_weight': 6, 'gamma': 1.0331012720332559e-06, 'lambda': 7.8243792452897
16e-07, 'alpha': 2.1222504936693533e-05}. Best is trial 25 with value: 22.8655666
38396167.
[I 2025-08-16 17:21:09,135] Trial 39 finished with value: 24.588352979513363 and
parameters: {'n_estimators': 884, 'learning_rate': 0.029462810250014734, 'max_dep
th': 9, 'subsample': 0.9096879368398562, 'colsample_bytree': 0.670481861778323,
'min_child_weight': 1, 'gamma': 0.0001387934163080436, 'lambda': 0.01140294207678
737, 'alpha': 1.0046544560006008e-07}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:21:14,897] Trial 40 finished with value: 23.526091491631874 and
parameters: {'n_estimators': 1410, 'learning_rate': 0.009588546574876566, 'max_de
pth': 7, 'subsample': 0.8734106152637644, 'colsample_bytree': 0.6367834642579603,
'min_child_weight': 7, 'gamma': 3.0283720895982918e-05, 'lambda': 0.0302669832317
```

```
57753, 'alpha': 0.0024145706804445297}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:21:16,136] Trial 41 finished with value: 23.097798248421366 and
parameters: {'n_estimators': 483, 'learning_rate': 0.012328787656255634, 'max_dep
th': 4, 'subsample': 0.7907322062381796, 'colsample_bytree': 0.796573391427735,
'min_child_weight': 5, 'gamma': 0.0015530559664514198, 'lambda': 0.00306623540804
88955, 'alpha': 6.693414043766107e-07}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:21:17,495] Trial 42 finished with value: 22.888488011787786 and
parameters: {'n_estimators': 452, 'learning_rate': 0.01168998176507176, 'max_dept
h': 5, 'subsample': 0.8485534131455198, 'colsample_bytree': 0.7760502678695318,
'min_child_weight': 4, 'gamma': 0.00014328209843373092, 'lambda': 0.0070051819026
16744, 'alpha': 1.0571617884702659e-08}. Best is trial 25 with value: 22.86556663
8396167.
[I 2025-08-16 17:21:19,237] Trial 43 finished with value: 23.66714058456839 and p
arameters: {'n_estimators': 539, 'learning_rate': 0.005771278285633397, 'max_dept
h': 5, 'subsample': 0.8440635877556617, 'colsample_bytree': 0.7823766560919495,
'min_child_weight': 3, 'gamma': 3.5949814125025114e-06, 'lambda': 0.6170311096334
252, 'alpha': 1.1973513564696392e-08}. Best is trial 25 with value: 22.8655666383
96167.
[I 2025-08-16 17:21:20,774] Trial 44 finished with value: 24.95684115882717 and p
arameters: {'n_estimators': 440, 'learning_rate': 0.0042665046477217364, 'max_dep
th': 6, 'subsample': 0.8892620291662007, 'colsample_bytree': 0.8398217270451752,
'min_child_weight': 4, 'gamma': 0.00010553139733941874, 'lambda': 0.0430740960781
4367, 'alpha': 2.5375646627424318e-08}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:21:25,475] Trial 45 finished with value: 23.037081680089646 and
parameters: {'n_estimators': 532, 'learning_rate': 0.008710528309877669, 'max_dep
th': 5, 'subsample': 0.8144055450689043, 'colsample_bytree': 0.7096551303278801,
'min_child_weight': 3, 'gamma': 3.885528006526359e-05, 'lambda': 1.12959151252308
99e-07, 'alpha': 1.527993898835526e-07}. Best is trial 25 with value: 22.86556663
8396167.
[I 2025-08-16 17:21:27,578] Trial 46 finished with value: 23.295716865267767 and
parameters: {'n_estimators': 631, 'learning_rate': 0.01691823843915408, 'max_dept
h': 6, 'subsample': 0.8461050112382167, 'colsample_bytree': 0.7618466710793005,
'min_child_weight': 4, 'gamma': 0.00022096606375191164, 'lambda': 0.9509007837218
14, 'alpha': 0.037171504298141374}. Best is trial 25 with value: 22.8655666383961
67.
[I 2025-08-16 17:21:28,905] Trial 47 finished with value: 34.19072177891497 and p
arameters: {'n_estimators': 446, 'learning_rate': 0.0010464027981396363, 'max_dep
th': 5, 'subsample': 0.948025042487932, 'colsample_bytree': 0.8930516093238743,
'min_child_weight': 6, 'gamma': 0.004855846929727968, 'lambda': 4.954326752857793
4e-05, 'alpha': 3.757871858760222e-08}. Best is trial 25 with value: 22.865566638
396167.
[I 2025-08-16 17:21:31,014] Trial 48 finished with value: 22.841772466626967 and
parameters: {'n_estimators': 530, 'learning_rate': 0.010199062373969763, 'max_dep
th': 7, 'subsample': 0.9020591029273664, 'colsample_bytree': 0.6661824717761999,
'min_child_weight': 6, 'gamma': 0.6631964931791021, 'lambda': 0.01154893844380962
8, 'alpha': 4.605981265856168e-05}. Best is trial 48 with value: 22.8417724666269
67.
[I 2025-08-16 17:21:34,808] Trial 49 finished with value: 28.33396971752624 and p
arameters: {'n_estimators': 774, 'learning_rate': 0.001398377385885757, 'max_dept
h': 8, 'subsample': 0.9262762466343012, 'colsample_bytree': 0.6730033425679911,
'min_child_weight': 6, 'gamma': 0.8458323169092313, 'lambda': 5.409955070750501e-
06, 'alpha': 5.923288080937225e-05}. Best is trial 48 with value: 22.841772466626
967.
```

```
--- Optimization Complete ---
Number of finished trials: 50
Best trial:
  Value (Validation RMSE): 22.8418
  Params:
    n_estimators: 530
    learning_rate: 0.010199062373969763
    max_depth: 7
    subsample: 0.9020591029273664
    colsample_bytree: 0.6661824717761999
    min_child_weight: 6
    gamma: 0.6631964931791021
    lambda: 0.011548938443809628
    alpha: 4.605981265856168e-05

--- Training Final Model with Optimal Parameters ---

--- Final Optimized Model Performance on Validation Set ---
MAE: 10.9706
RMSE: 18.9220
R^2:  0.7702

--- Final Optimized Model Performance on Test Set ---
MAE: 15.8385
RMSE: 26.4637
R^2:  0.6351

--- Saving the Optimized XGBoost Model ---
Optimized model saved to '/content/keras_tuner_dir/ev_demand_forecasting/ev_deman
d_optimized_xgboost_model.json'
```

Final Plot Customisation for Optimised XGBoost Model

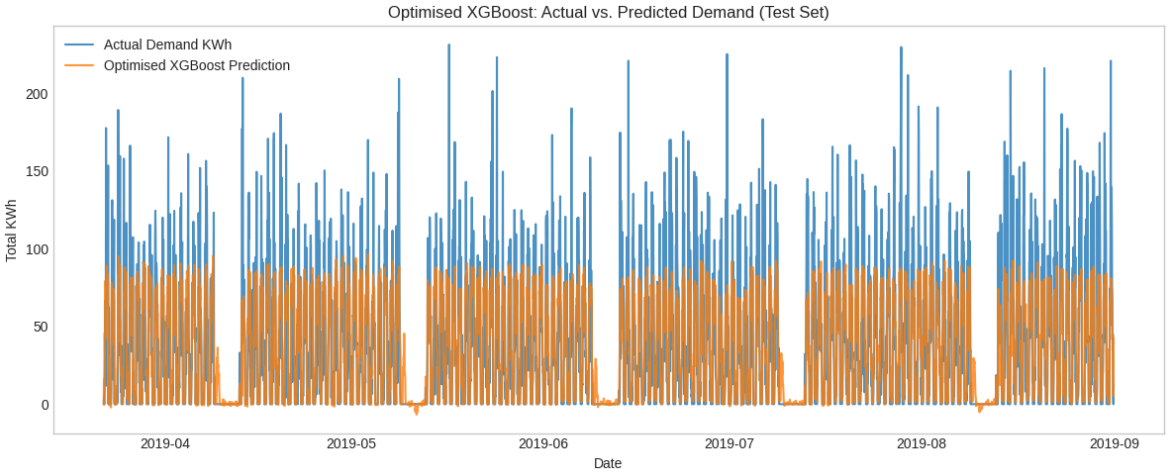
I am now applying the final formatting to the visualisation. I set the main title and the labels for the x and y axes to ensure the plot is clearly understood. A legend is also added to differentiate between the actual and predicted data. Finally, I save the completed chart as a high-resolution image for my dissertation before displaying it.

In [123…

```python
# Plot Actual vs. Predicted on Test Set ---
print("\n--- Generating Actual vs. Predicted Plot ---")

# Create a pandas Series for the predictions with the same index as y_test
y_test_pred_series = pd.Series(y_test_pred_optimized, index=y_test.index)

plt.figure(figsize=(12, 5))

# Plot the actual values from the test set
plt.plot(y_test, label='Actual Demand KWh', alpha=0.8)

# Plot the predictions from the final XGBoost model
plt.plot(y_test_pred_series, label='Optimised XGBoost Prediction', alpha=0.8)

# --- Formatting ---
plt.title('Optimised XGBoost: Actual vs. Predicted Demand (Test Set)')
plt.xlabel('Date')
plt.ylabel('Total KWh')
```

```
plt.legend()
plt.tight_layout()
plt.grid(False)

# --- Save the Figure ---
plt.savefig("XGBoost_final_timeseries_visualisation_no_grid.png", dpi=1200)

plt.show()
```

```
--- Generating Actual vs. Predicted Plot ---
```



Analysing Feature Importance

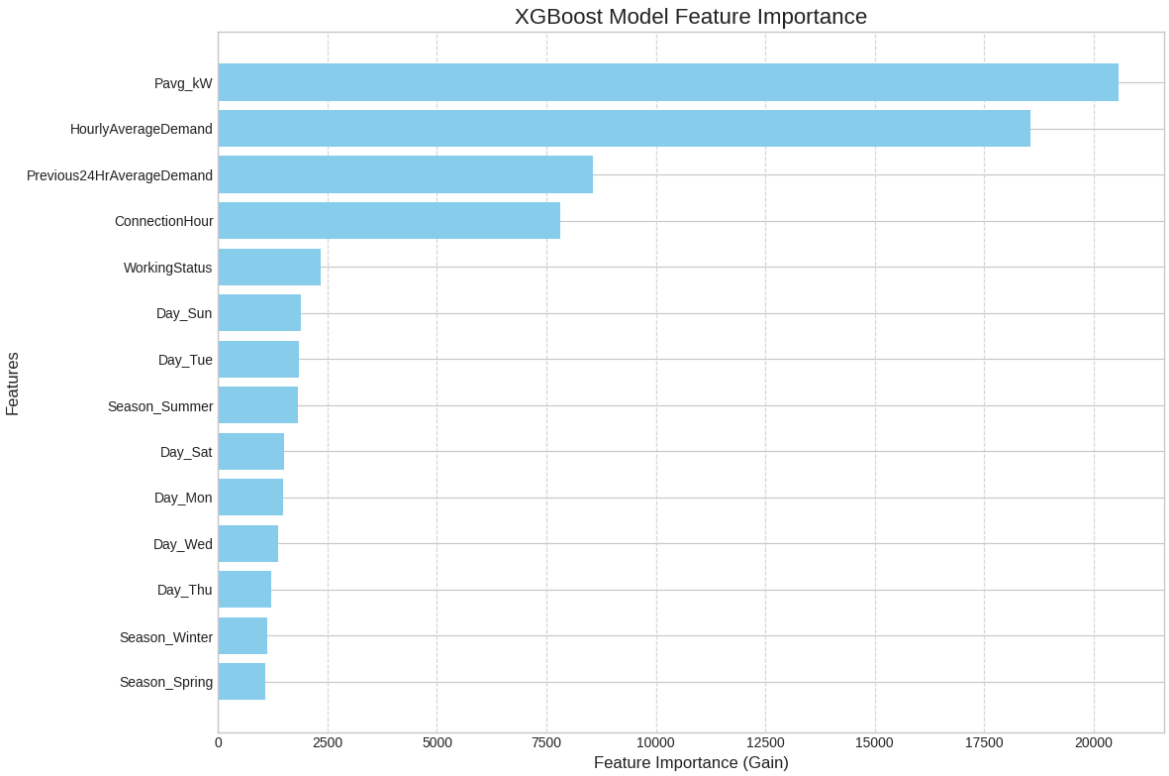
I am creating a function to analyse which features are most influential in the optimised XGBoost model's predictions. The function first loads the saved model from its file. It then extracts the feature importance scores, using the 'gain' metric which measures each feature's relative contribution to the model's performance. These scores are then sorted and used to generate a horizontal bar chart, providing a clear, ranked visualisation of which data points the model relies on most. Finally, the plot is saved as a high-resolution image for my dissertation. This entire process is wrapped in a main execution block to ensure it runs cleanly as a script.

In [125…

```
import os

# Define the path to your saved, optimized XGBoost model
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"
MODEL_PATH = os.path.join(MODELS_DIR, 'ev_demand_optimized_xgboost_model.json')

# Define where to save the output plot
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)

# Load model and extract importance

def plot_feature_importance(model_path, save_dir):
    """
    Loads a trained XGBoost model and plots its feature importance.
    """
    print(f"--- Loading model from: {model_path} ---")

    try:
```

```
        # Load the trained XGBoost model
        model = xgb.XGBRegressor()
        model.load_model(model_path)
        print(" Model loaded successfully.")

        # Get feature importance scores. 'gain' is a good metric as it represent
        # the relative contribution of the feature to the model.
        importance_scores = model.get_booster().get_score(importance_type='gain'

        if not importance_scores:
            print(" Error: Could not retrieve feature importance scores from the
            return

        # Create a pandas DataFrame for easier sorting and plotting
        importance_df = pd.DataFrame({
            'Feature': list(importance_scores.keys()),
            'Importance': list(importance_scores.values())
        }).sort_values(by='Importance', ascending=True)


        # Visualisation

        print("\n--- Generating Feature Importance Plot ---")

        plt.style.use('seaborn-v0_8-whitegrid')
        plt.figure(figsize=(12, 8))

        # Create a horizontal bar plot
        plt.barh(importance_df['Feature'], importance_df['Importance'], color='s

        plt.xlabel("Feature Importance (Gain)", fontsize=12)
        plt.ylabel("Features", fontsize=12)
        plt.title("XGBoost Model Feature Importance", fontsize=16)
        plt.grid(axis='x', linestyle='--', alpha=0.7)
        plt.tight_layout()

        # Save the plot
        save_path = os.path.join(save_dir, "xgboost_feature_importance.png")
        plt.savefig(save_path, dpi=1200, bbox_inches='tight')
        plt.show()

        print(f"\n Feature importance plot saved to: {save_path}")

    except FileNotFoundError:
        print(f" CRITICAL ERROR: Model file not found at '{model_path}'.")
        print("Please ensure you have run the Optuna optimization pipeline to sa
    except Exception as e:
        print(f"An unexpected error occurred: {e}")

# Main execution

if __name__ == "__main__":
    plot_feature_importance(MODEL_PATH, SAVE_DIR)
```

```
--- Loading model from: /content/keras_tuner_dir/ev_demand_forecasting/ev_demand_
optimized_xgboost_model.json ---
 Model loaded successfully.

--- Generating Feature Importance Plot ---
```




```
Feature importance plot saved to: project_visualizations/xgboost_feature_importa
nce.png
```

Final Visualisation Pipeline

To produce the final visual evidence for my dissertation, I have created a comprehensive plotting pipeline.

First, I define a function to load my two best-performing models: the optimised XGBoost and the tuned CNN-LSTM. This keeps the main script clean and organised.

Next, I create two powerful and reusable plotting functions. The first, plot_day_comparison_optimized, is designed to generate a detailed 24-hour comparison for any given date, plotting the actual demand against the predictions from my best models. The second function, plot_week_comparison_optimized, extends this capability to visualise performance over an entire seven-day period.

Finally, in the main execution block, I use these functions to systematically generate all the key comparison charts for my analysis. I select specific dates representing a typical working day, a weekend, a bank holiday, and also special periods like the Easter and Christmas weeks. This allows me to create a robust visual comparison of how my final models perform across a variety of real-world demand scenarios.

In [97]:
```
# Visualization and Comparison Pipeline ---
print("\n--- Generating Comparison Plots for Optimized Model ---")

def load_comparison_models(models_dir='/content/keras_tuner_dir/ev_demand_foreca
    """Loads all models needed for final comparison plots."""
    models = {}
    print("--- Loading Models for Comparison ---")
    try:
```

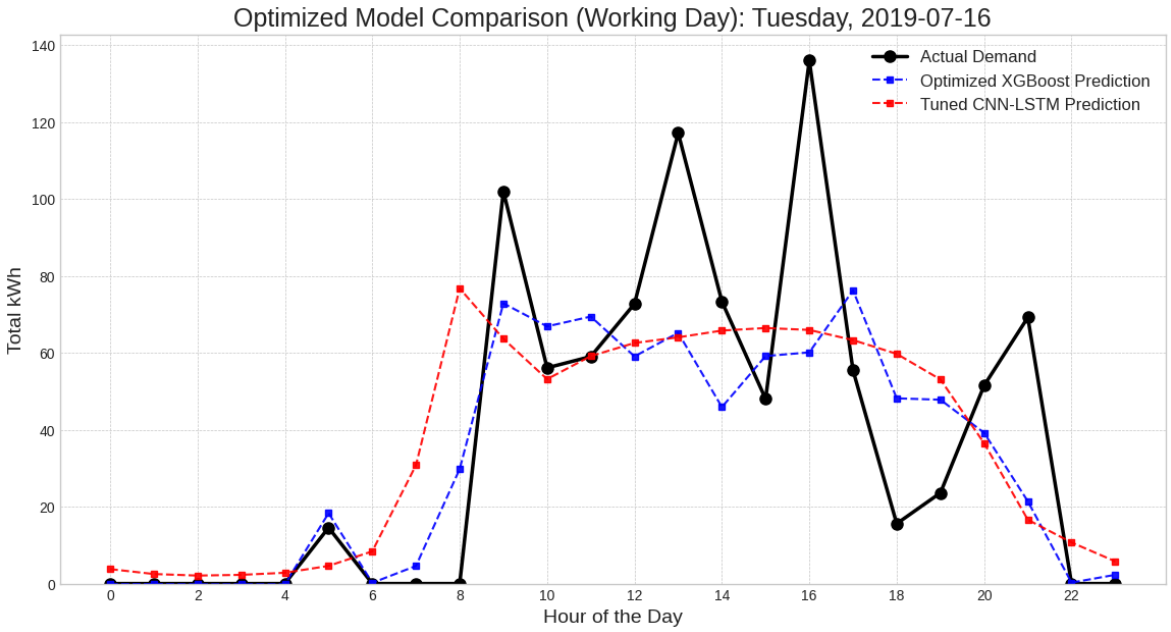
```
        # Load the newly optimized XGBoost model
        xgb_opt_model = xgb.XGBRegressor()
        xgb_opt_model.load_model(os.path.join(models_dir, 'ev_demand_optimized_x
        models['Optimized XGBoost'] = xgb_opt_model
        print("Optimized XGBoost model loaded.")

        # Load the tuned CNN-LSTM model
        models['Tuned CNN-LSTM'] = tf.keras.models.load_model(os.path.join(model
        print("Tuned CNN-LSTM model loaded.")

    except Exception as e:
        print(f"An error occurred while loading models for plotting: {e}")
        return None
    return models

def plot_day_comparison_optimized(target_date_str, models, data_dict, title_pref
    """Generates and plots predictions from optimized models for a single day.""
    if data_split == 'validation':
        X_data, y_data, X_scaled = data_dict['X_val'], data_dict['y_val'], data_
    else:
        X_data, y_data, X_scaled = data_dict['X_test'], data_dict['y_test'], dat

    scaler_y, window_size = data_dict['scaler_y'], 24

    target_date = pd.to_datetime(target_date_str)
    day_mask = (X_data.index.date == target_date.date())

    if not day_mask.any():
        print(f"Error: No data for {target_date_str} in '{data_split}' set.")
        return

    y_actual_day, X_day = y_data[day_mask], X_data[day_mask]

    try:
        start_idx = np.where(X_data.index == y_actual_day.index[0])[0][0]
    except IndexError:
        print(f"Error: Could not find start of day {target_date_str} in index.")
        return

    X_day_seq = np.array([X_scaled[start_idx + i - window_size : start_idx + i]

    predictions = {}
    for name, model in models.items():
        print(f"Generating predictions for {name}...")
        if "CNN-LSTM" in name:
            pred_scaled = model.predict(X_day_seq, verbose=0)
            predictions[name] = scaler_y.inverse_transform(pred_scaled).flatten(
        else:
            predictions[name] = model.predict(X_day)

    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(14, 7))
    plt.plot(y_actual_day.values, 'o-', label='Actual Demand', color='black', li

    colors = {'Optimized XGBoost': 'blue', 'Tuned CNN-LSTM': 'red'}
    for name, pred_values in predictions.items():
        plt.plot(pred_values, 's--', label=f'{name} Prediction', color=colors.ge

    plt.title(f'{title_prefix}: {target_date.strftime("%A, %Y-%m-%d")}', fontsiz
    plt.xlabel('Hour of the Day', fontsize=14)
```

```
    plt.ylabel('Total kWh', fontsize=14)
    plt.xticks(ticks=np.arange(0, 24, 2))
    plt.legend(fontsize=12)
    plt.ylim(bottom=0)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    filename = f"{title_prefix.replace(' ', '_')}_{target_date.strftime('%Y%m%d'
    plt.savefig(filename, dpi=1200, bbox_inches='tight')
    print(f"Plot saved as: {filename}")
    plt.show()

def plot_week_comparison_optimized(start_date_str, models, data_dict, title_pref
    """Generates and plots predictions from optimized models for a single week."
    if data_split == 'validation':
        X_data, y_data, X_scaled = data_dict['X_val'], data_dict['y_val'], data_
    else:
        X_data, y_data, X_scaled = data_dict['X_test'], data_dict['y_test'], dat

    scaler_y, window_size = data_dict['scaler_y'], 24

    start_date = pd.to_datetime(start_date_str)
    end_date = start_date + pd.Timedelta(days=6)
    week_mask = (X_data.index.date >= start_date.date()) & (X_data.index.date <=

    if not week_mask.any():
        print(f"Error: No data for week starting {start_date_str} in '{data_spli
        return

    y_actual_week, X_week = y_data[week_mask], X_data[week_mask]

    try:
        start_idx = np.where(X_data.index == y_actual_week.index[0])[0][0]
    except IndexError:
        print(f"Error: Could not find start of week {start_date_str} in index.")
        return

    X_week_seq = np.array([X_scaled[start_idx + i - window_size : start_idx + i]

    predictions = {}
    for name, model in models.items():
        print(f"Generating predictions for {name}...")
        if "CNN-LSTM" in name:
            pred_scaled = model.predict(X_week_seq, verbose=0)
            predictions[name] = scaler_y.inverse_transform(pred_scaled).flatten(
        else:
            predictions[name] = model.predict(X_week)

    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(18, 8))
    plt.plot(y_actual_week.values, 'o-', label='Actual Demand', color='black', l

    colors = {'Optimized XGBoost': 'blue', 'Tuned CNN-LSTM': 'red'}
    for name, pred_values in predictions.items():
        plt.plot(pred_values, 's--', label=f'{name} Prediction', color=colors.ge

    plt.title(f'{title_prefix}: Week of {start_date.strftime("%Y-%m-%d")} to {en
    plt.xlabel('Hour of the Week (0-167)', fontsize=14)
    plt.ylabel('Total kWh', fontsize=14)
    plt.xticks(ticks=np.arange(0, 168, 24), labels=['Mon', 'Tue', 'Wed', 'Thu',
    plt.legend(fontsize=12)
    plt.ylim(bottom=0)
```

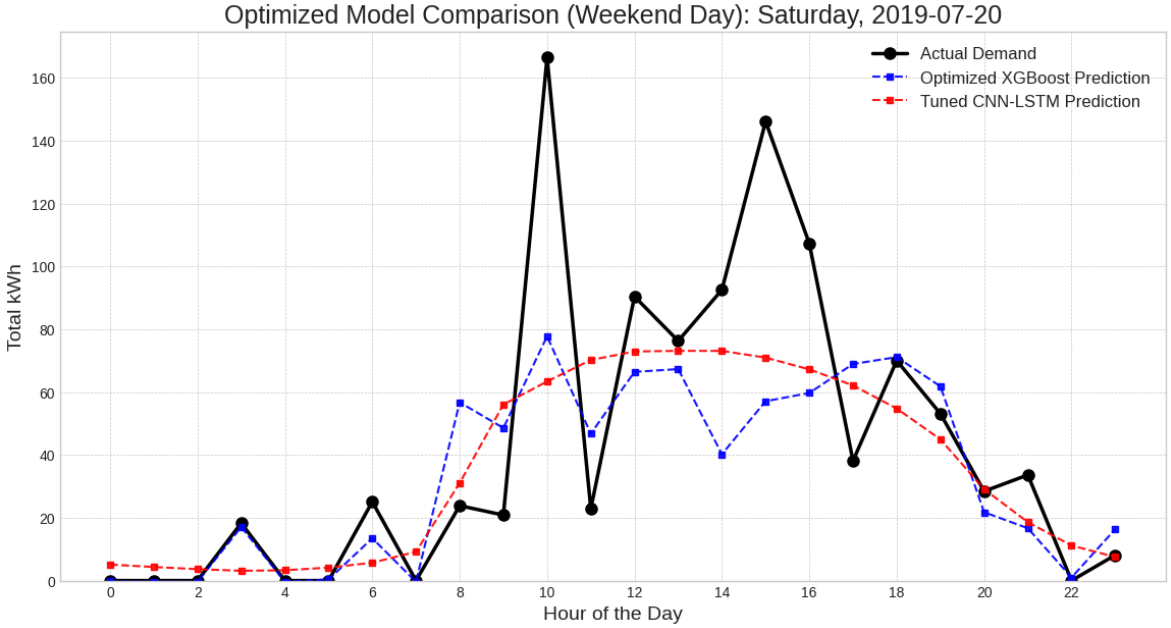
```
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    # Save the plot
    filename = f"{title_prefix.replace(' ', '_')}_{start_date.strftime('%Y%m%d')
    plt.savefig(filename, dpi=1200, bbox_inches='tight')
    print(f"Plot saved as: {filename}")
    plt.show()

try:
    comparison_models = load_comparison_models()
    data_for_plotting = {
        'X_test': X_test, 'y_test': y_test, 'X_test_scaled': X_test_scaled,
        'X_val': X_val, 'y_val': y_val, 'X_val_scaled': X_val_scaled,
        'scaler_y': scaler_y
    }

    if comparison_models:
        # Generate Daily Comparison Plots ---
        print("\n--- Generating Daily Comparison Plots ---")
        plot_day_comparison_optimized(
            target_date_str='2019-07-16',
            models=comparison_models,
            data_dict=data_for_plotting,
            title_prefix="Optimized Model Comparison (Working Day)",
            data_split='test'
        )
        plot_day_comparison_optimized(
            target_date_str='2019-07-20',
            models=comparison_models,
            data_dict=data_for_plotting,
            title_prefix="Optimized Model Comparison (Weekend Day)",
            data_split='test'
        )
        plot_day_comparison_optimized(
            target_date_str='2019-08-26',
            models=comparison_models,
            data_dict=data_for_plotting,
            title_prefix="Optimized Model Comparison (Bank Holiday)",
            data_split='test'
        )

        # Generate Weekly Comparison Plots ---
        print("\n--- Generating Weekly Comparison Plots ---")
        plot_week_comparison_optimized(
            start_date_str='2019-07-15',
            models=comparison_models,
            data_dict=data_for_plotting,
            title_prefix="Optimized Model Comparison (Typical Week)",
            data_split='test'
        )
        plot_week_comparison_optimized(
            start_date_str='2019-04-19',
            models=comparison_models,
            data_dict=data_for_plotting,
            title_prefix="Optimized Model Comparison (Easter Wek)",
            data_split='test'
        )
        plot_week_comparison_optimized(
            start_date_str='2018-12-24',
            models=comparison_models,
            data_dict=data_for_plotting,
```

```
            title_prefix="Optimized Model Comparison (Christmas Week)",
            data_split='validation'
        )
except NameError as e:
    print(f"\nPlotting failed: {e}. Ensure all data variables (X_train, y_val, e
```

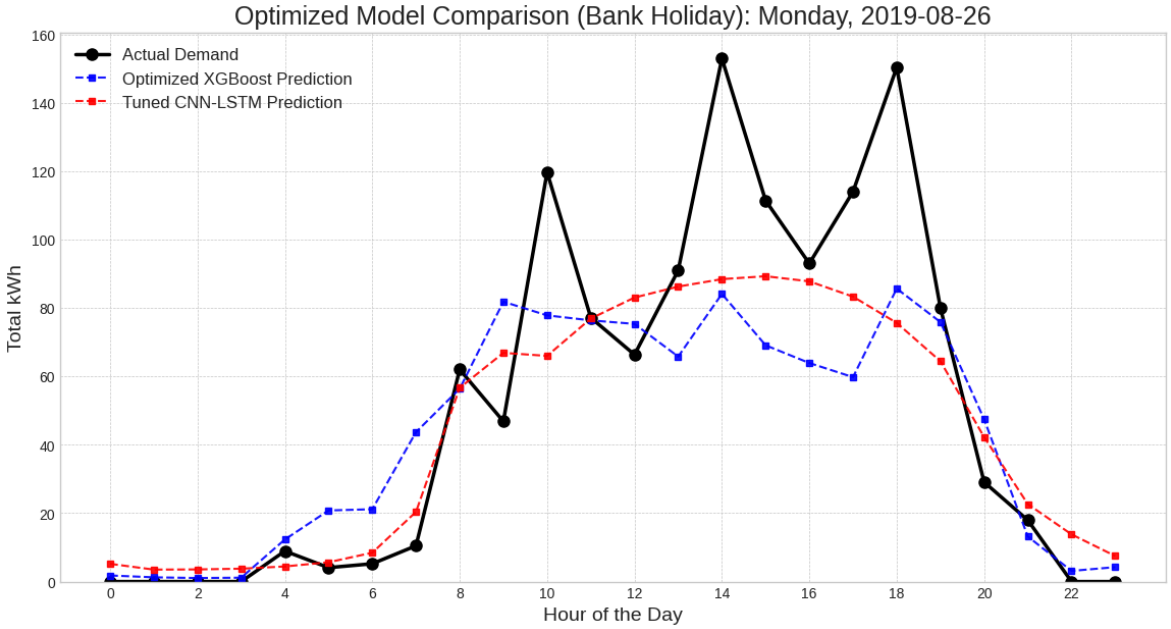
```
--- Generating Comparison Plots for Optimized Model ---
--- Loading Models for Comparison ---
Optimized XGBoost model loaded.
Tuned CNN-LSTM model loaded.

--- Generating Daily Comparison Plots ---
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Working_Day)_20190716.png
```



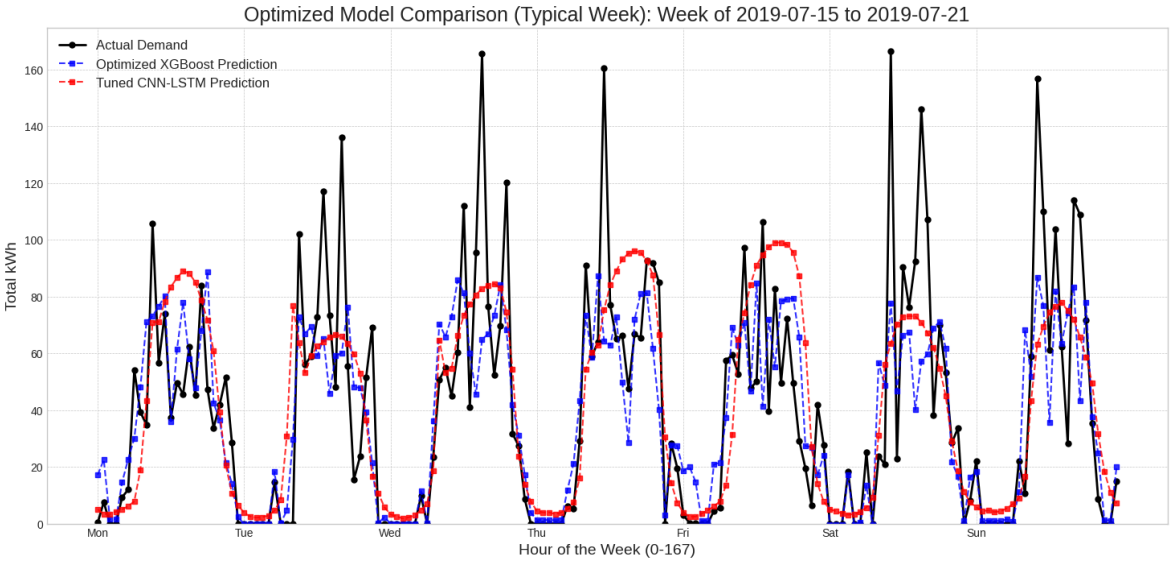

```
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Weekend_Day)_20190720.png
```



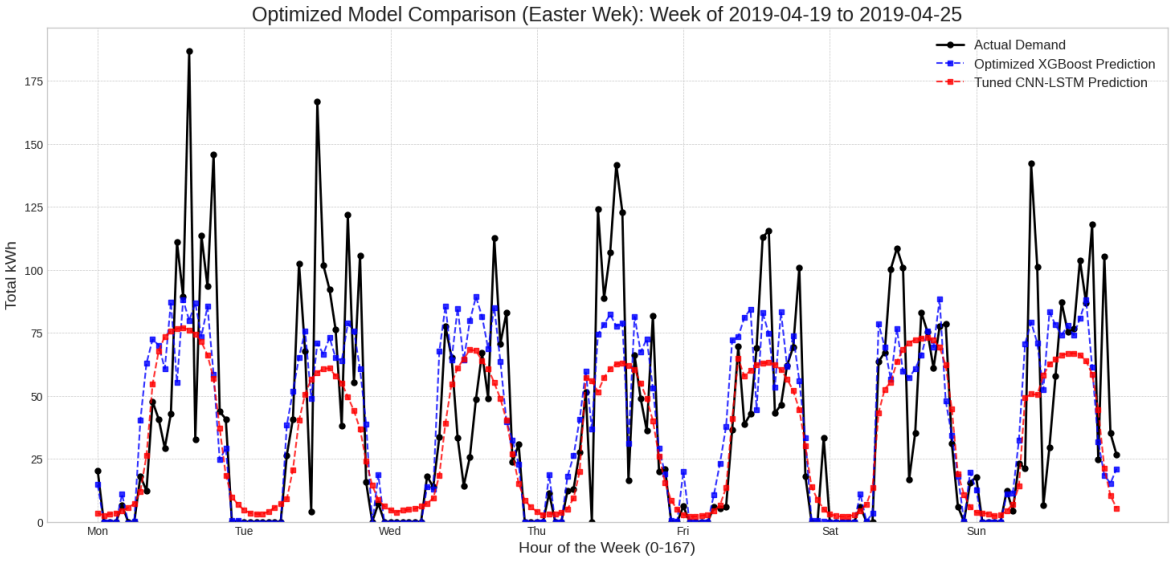

```
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Bank_Holiday)_20190826.png
```



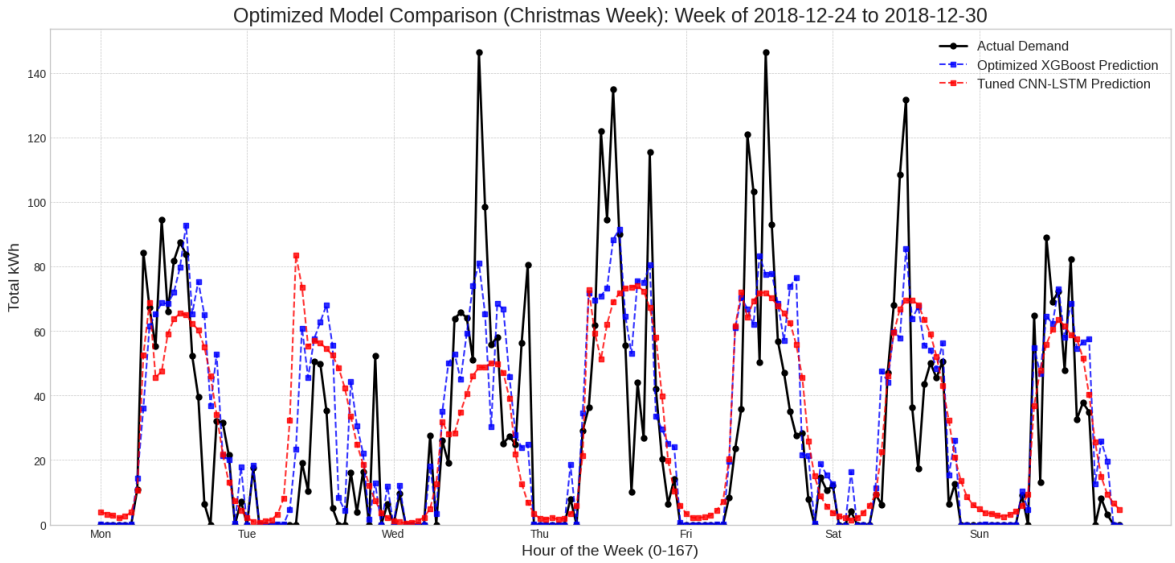

```
--- Generating Weekly Comparison Plots ---
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Typical_Week)_20190715.png
```




```
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Easter_Wek)_20190419.png
```



```
Generating predictions for Optimized XGBoost...
Generating predictions for Tuned CNN-LSTM...
Plot saved as: Optimized_Model_Comparison_(Christmas_Week)_20181224.png
```



Quantifying Forecast Uncertainty

To create a forecast that accounts for real-world uncertainty, I am now moving beyond single-point predictions. Instead of one model that predicts the single most likely outcome, I am training a set of XGBoost models using a technique called quantile regression.

I have defined a pipeline that first trains three separate models, each one optimised to predict a specific percentile of the energy demand: the 10th percentile (a low-demand scenario), the 50th percentile (the median or most likely outcome), and the 90th percentile (a high-demand scenario).

Once these models are trained, I use them to generate a 24-hour forecast for a specific day. The final visualisation plots the actual demand against the median forecast, and critically, it includes a shaded area between the 10th and 90th percentile predictions. This shaded region represents the prediction interval, providing a clear, quantitative measure of the forecast's uncertainty.

In [127…

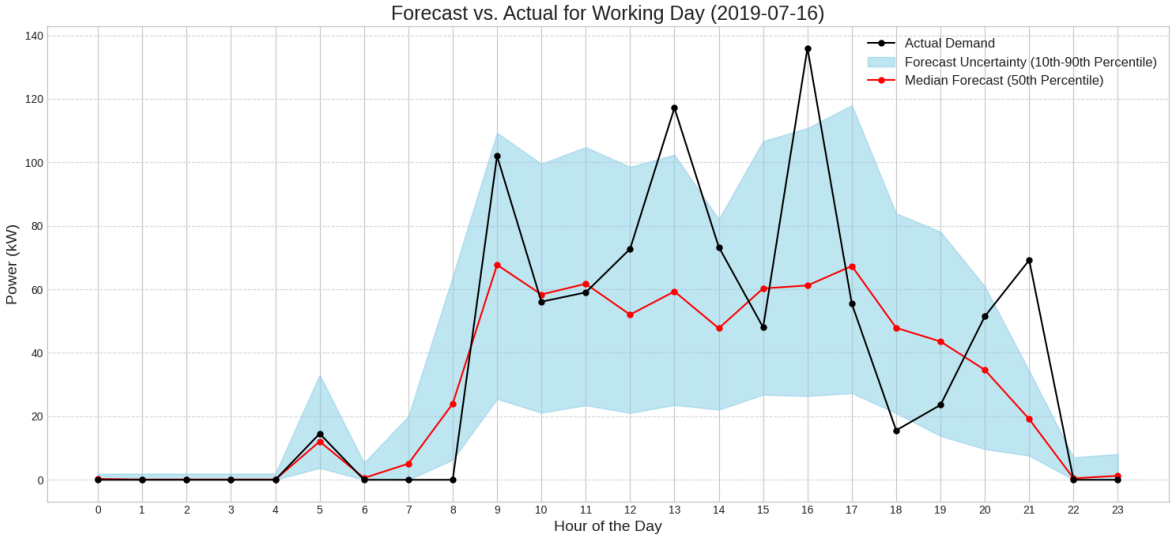
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import os
from datetime import datetime, timedelta


# This assumes 'best_params', 'X_train', 'y_train', 'X_val', 'y_val', 'X_test',
# and 'y_test' are already defined in your notebook's memory.
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"
QUANTILES_TO_TRAIN = [0.10, 0.50, 0.90] # Lower bound, median, upper bound

# Model Training
def train_and_save_quantile_models(params, X_train, y_train, X_val, y_val, quant
    """Trains and saves XGBoost models for specified quantiles."""
```

```
    print("--- Training Quantile Regression Models ---")
    base_params = params.copy()

    if not os.path.exists(models_dir):
        os.makedirs(models_dir)

    for q in quantiles:
        print(f"Training model for quantile: {q:.2f}")
        base_params['objective'] = 'reg:quantileerror'
        base_params['quantile_alpha'] = q

        model = xgb.XGBRegressor(**base_params)

        X_train_full = pd.concat([X_train, X_val])
        y_train_full = pd.concat([y_train, y_val])

        model.fit(X_train_full, y_train_full, verbose=False)

        model_path = os.path.join(models_dir, f'ev_demand_xgb_quantile_{int(q*10
        model.save_model(model_path)
        print(f"Model for quantile {q:.2f} saved to {model_path}")

# Forecast generation
def generate_forecasts(models_dir, data_source, start_date_str, end_date_str, qu
    """
    Loads pre-trained models and generates forecasts for a specified date range.
    """
    print(f"\n🔮 Generating forecasts from {start_date_str} to {end_date_str}..

    start_date = pd.to_datetime(start_date_str)
    end_date = pd.to_datetime(end_date_str)
    period_mask = (data_source.index.date >= start_date.date()) & (data_source.i

    if not period_mask.any():
        print(f"Error: No data found for the specified period in the data source
        return None

    X_period = data_source[period_mask]

    forecasts = {}
    for q in quantiles:
        try:
            model_path = os.path.join(models_dir, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
        except Exception as e:
            raise FileNotFoundError(f"Error loading model for quantile {q}: {e}"

        forecasted_demand = model.predict(X_period)
        forecasted_demand = np.maximum(forecasted_demand, 0)
        forecasts[f'q_{q}'] = forecasted_demand

    return pd.DataFrame(forecasts, index=X_period.index)

# Define visualisations
def visualize_forecast_vs_actual(forecast_df, actual_series, title):
    """
    Generates a plot comparing the forecast (with uncertainty) against actual de
    """
    print(f"--- Visualizing: {title} ---")
```

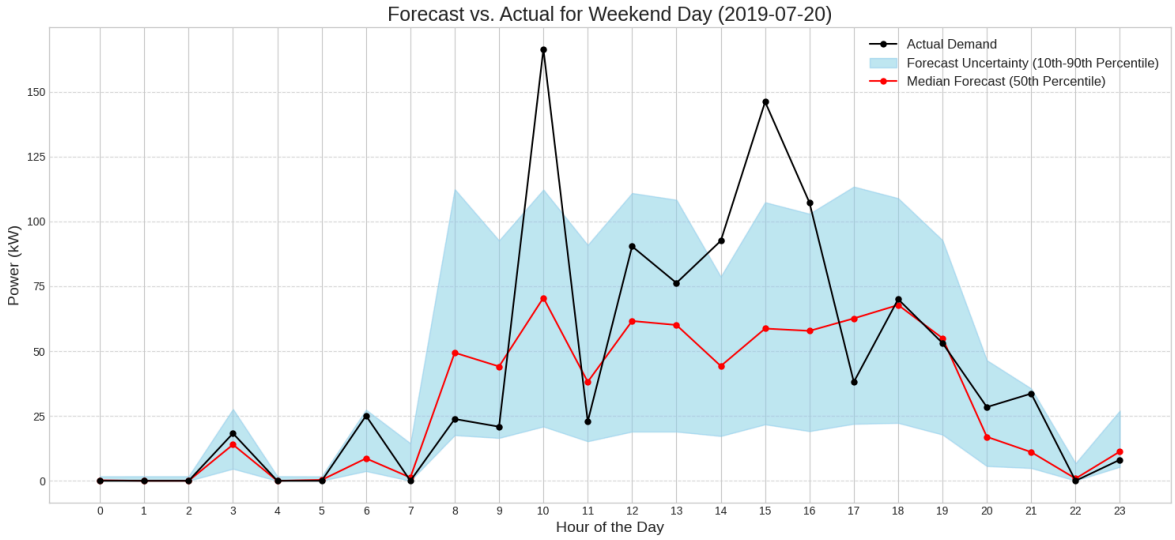
```
    actual_demand = actual_series.loc[forecast_df.index]

    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(15, 7))

    time_index = np.arange(len(forecast_df))

    # Plot the actual demand
    plt.plot(time_index, actual_demand.values, 'o-', color='black', label='Actua

    # Plot the uncertainty interval
    plt.fill_between(
        time_index,
        forecast_df['q_0.1'],
        forecast_df['q_0.9'],
        color='skyblue',
        alpha=0.5,
        label='Forecast Uncertainty (10th-90th Percentile)'
    )

    # Plot the median forecast
    plt.plot(time_index, forecast_df['q_0.5'], 'o-', color='red', label='Median

    plt.title(title, fontsize=18)
    plt.xlabel('Time', fontsize=14)
    plt.ylabel('Power (kW)', fontsize=14)

    # Adjust x-axis ticks based on the period length
    if len(time_index) <= 24: # Daily plot
        plt.xticks(time_index)
        plt.xlabel('Hour of the Day', fontsize=14)
    elif len(time_index) <= 24 * 7: # Weekly plot
        num_days = int(np.ceil(len(time_index)/24))
        plt.xticks(np.arange(0, len(time_index), 24), [f'Day {i+1}' for i in ran

    plt.legend(fontsize=12)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.tight_layout()
    filename = f"{title.replace(' ', '_').replace('(', '').replace(')', '')}.png
    plt.savefig(filename, dpi=1200, bbox_inches='tight')
    print(f"Plot saved as: {filename}")
    plt.show()

# Main execution
if __name__ == "__main__":
    try:
        # Step A: Train the quantile models (run this once)
        # Assumes 'best_params' from Optuna study is available
        train_and_save_quantile_models(best_params, X_train, y_train, X_val, y_v

        # --- Scenario 1: Typical Working Day ---
        working_day = '2019-07-16'
        forecast_working_day = generate_forecasts(MODELS_DIR, X_test, working_da
        if forecast_working_day is not None:
            visualize_forecast_vs_actual(forecast_working_day, y_test, f"Forecas

        # --- Scenario 2: Typical Weekend Day ---
        weekend_day = '2019-07-20'
        forecast_weekend_day = generate_forecasts(MODELS_DIR, X_test, weekend_da
```

```
        if forecast_weekend_day is not None:
            visualize_forecast_vs_actual(forecast_weekend_day, y_test, f"Forecas

        # --- Scenario 3: Bank Holiday ---
        bank_holiday = '2019-08-26' # Summer Bank Holiday
        forecast_bank_holiday = generate_forecasts(MODELS_DIR, X_test, bank_holi
        if forecast_bank_holiday is not None:
            visualize_forecast_vs_actual(forecast_bank_holiday, y_test, f"Foreca

        # --- Scenario 4: Typical Week ---
        week_start = '2019-07-15'
        week_end = '2019-07-21'
        forecast_week = generate_forecasts(MODELS_DIR, X_test, week_start, week_
        if forecast_week is not None:
            visualize_forecast_vs_actual(forecast_week, y_test, f"Forecast vs. A

    except NameError as e:
        print(f"\n EXECUTION FAILED: A required variable is not defined: {e}")
        print("Please ensure 'best_params', 'X_train', 'y_train', 'X_val', 'y_va
    except Exception as e:
        print(f"\n An unexpected error occurred: {e}")
```

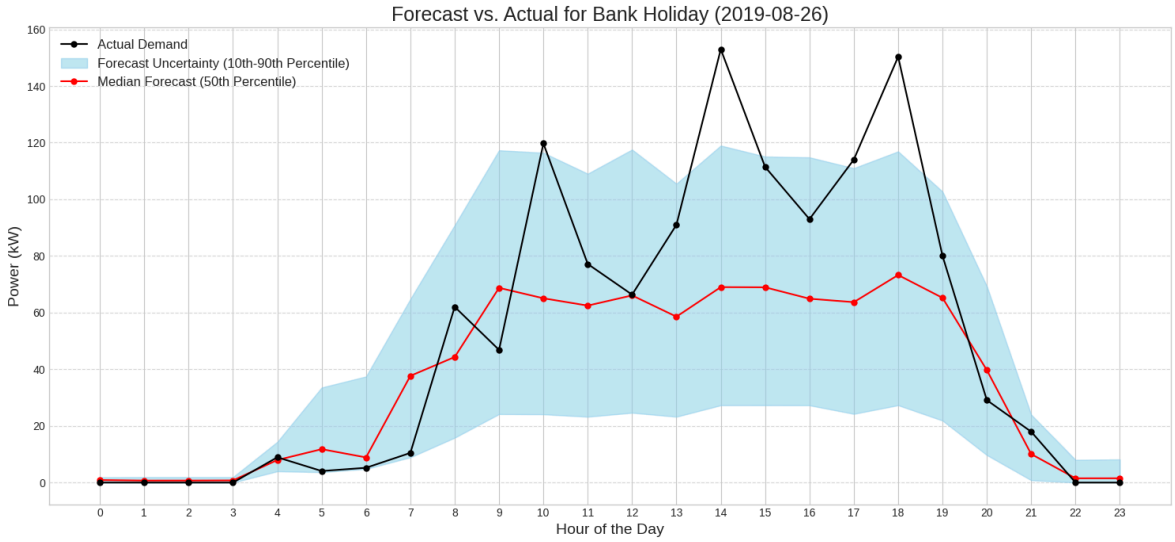
```
--- Training Quantile Regression Models ---
Training model for quantile: 0.10
Model for quantile 0.10 saved to /content/keras_tuner_dir/ev_demand_forecasting/e
v_demand_xgb_quantile_10.json
Training model for quantile: 0.50
Model for quantile 0.50 saved to /content/keras_tuner_dir/ev_demand_forecasting/e
v_demand_xgb_quantile_50.json
Training model for quantile: 0.90
Model for quantile 0.90 saved to /content/keras_tuner_dir/ev_demand_forecasting/e
v_demand_xgb_quantile_90.json

🔮 Generating forecasts from 2019-07-16 to 2019-07-16...
--- Visualizing: Forecast vs. Actual for Working Day (2019-07-16) ---
Plot saved as: Forecast_vs._Actual_for_Working_Day_2019-07-16.png
```



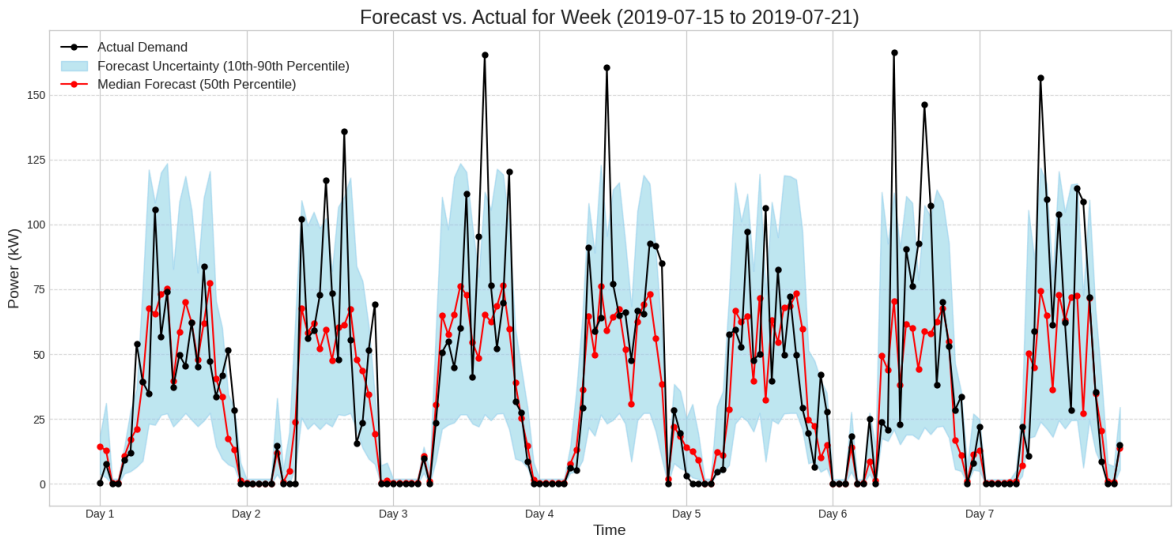

```
🔮 Generating forecasts from 2019-07-20 to 2019-07-20...
--- Visualizing: Forecast vs. Actual for Weekend Day (2019-07-20) ---
Plot saved as: Forecast_vs._Actual_for_Weekend_Day_2019-07-20.png
```




```
Generating forecasts from 2019-08-26 to 2019-08-26...
--- Visualizing: Forecast vs. Actual for Bank Holiday (2019-08-26) ---
Plot saved as: Forecast_vs._Actual_for_Bank_Holiday_2019-08-26.png
```




```
Generating forecasts from 2019-07-15 to 2019-07-21...
--- Visualizing: Forecast vs. Actual for Week (2019-07-15 to 2019-07-21) ---
Plot saved as: Forecast_vs._Actual_for_Week_2019-07-15_to_2019-07-21.png
```



OPTIMISATION

Implementing the Cost Optimisation

This script brings together the forecasting and optimisation stages to find the most cost-effective 24-hour charging schedule.

First, I define the core optimisation function. This function builds a linear programming model whose objective is to minimise the total electricity cost, which is a combination of the energy consumed (priced at a time-of-use tariff) and a demand charge for the highest power peak. I give this model a set of rules, or constraints, it must follow: it has to deliver the total amount of energy predicted by my median forecast, it cannot exceed the hourly demand predicted by my 90th-percentile forecast, and it must respect the physical ramp-rate limits of the charging hardware.

In the main part of the script, I define the economic parameters, such as the hourly electricity prices and the demand charge. I then run the entire pipeline for a sample day:

- I generate the demand forecast with its uncertainty bounds.
- I run the optimisation twice: once with the ramp-rate limits applied and once without, to analyse the impact of this constraint.
- Finally, I produce a visualisation that plots the original forecast against the two optimised schedules, all overlaid on a graph of the electricity prices. This clearly demonstrates how the optimiser strategically shifts energy usage to cheaper off-peak hours while staying within all the required operational limits. I also calculate and print the precise cost savings achieved by each optimisation strategy compared to a naive, un-optimised approach.

In [132…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import pulp


# --- Prerequisites ---
# This code assumes the following variables from your notebook are available:
# - X_test, y_test: The test data split.
# It also assumes you have saved your quantile regression models.

# --- 1. Generate Forecast with Uncertainty ---
def generate_quantile_forecasts(models_dir, data_source, target_date_str, quanti
    """
    Loads the optimized quantile models and generates a 24-hour forecast
    for a specific target day, including prediction intervals.
    """
    print(f"--- Generating quantile forecasts for {target_date_str} ---")
    forecasts = {}
    target_date = pd.to_datetime(target_date_str)
    day_mask = (data_source.index.date == target_date.date())

    if not day_mask.any():
```

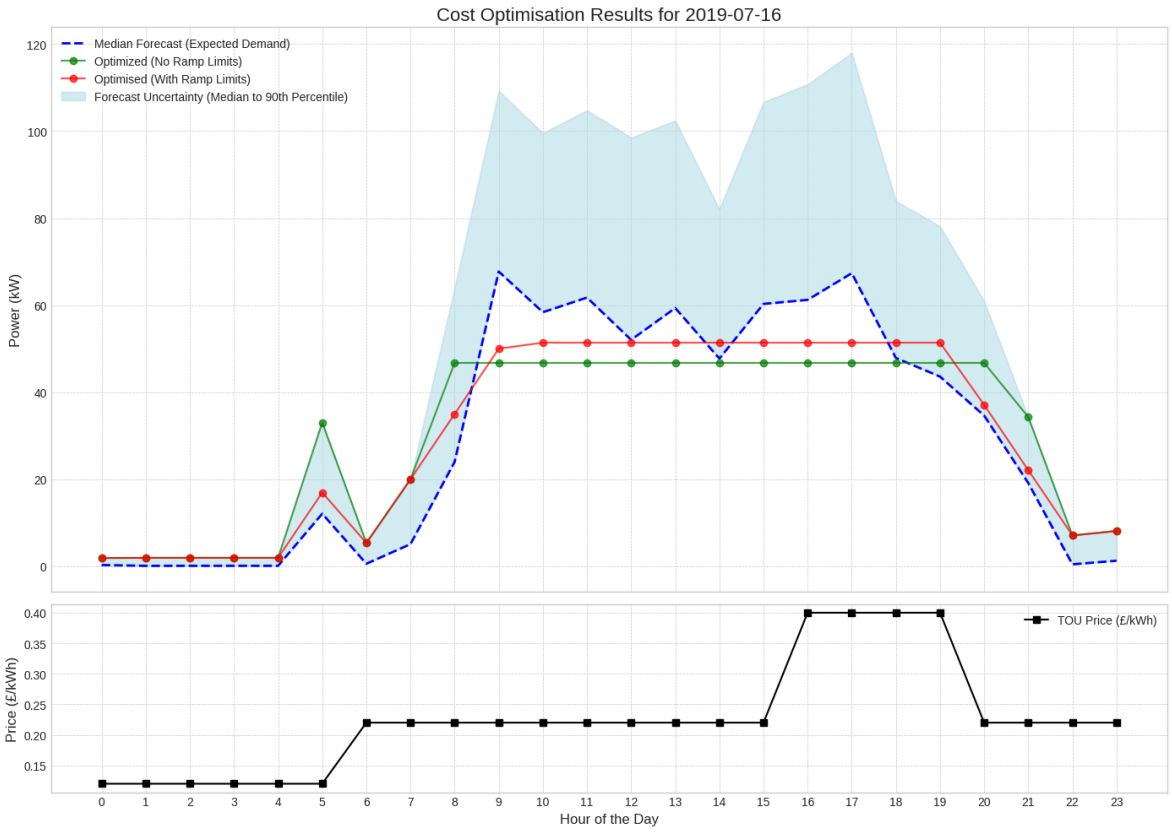
```
        print(f"Error: No data found for {target_date_str} in the data source.")
        return None

    X_day = data_source[day_mask]

    for q in quantiles:
        try:
            model_path = os.path.join(models_dir, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
        except Exception as e:
            print(f"Error loading model for quantile {q}: {e}")
            return None

        forecasted_demand = model.predict(X_day)
        forecasted_demand[forecasted_demand < 0] = 0
        forecasts[f'q_{q}'] = forecasted_demand

    return pd.DataFrame(forecasts, index=X_day.index)

# --- 2. Corrected Optimization Function with Upper Bounds ---
def solve_cost_optimization(tou_prices, demand_charge_price, max_ramp_rate,
                            total_energy_constraint, num_intervals, upper_bounds
    """
    Formulates and solves the cost optimization problem, now including the
    hourly upper bound constraint for a more realistic scenario.
    """
    print(f"\n--- Formulating optimization (Ramp Rate <= {max_ramp_rate} kW/hr)

    prob = pulp.LpProblem("Cost_Minimization", pulp.LpMinimize)

    # Variables
    scheduled_power = pulp.LpVariable.dicts("ScheduledPower", range(num_interval
    peak_power = pulp.LpVariable("PeakPower", lowBound=0, cat='Continuous')

    # Objective
    energy_cost = pulp.lpSum(scheduled_power[i] * tou_prices[i] for i in range(n
    demand_charge = peak_power * demand_charge_price
    prob += energy_cost + demand_charge, "Minimize_Total_Cost"

    # Constraints
    prob += pulp.lpSum(scheduled_power[i] for i in range(num_intervals)) >= tota

    for i in range(num_intervals):
        prob += peak_power >= scheduled_power[i], f"Peak_Constraint_{i}"

    # Ramp constraints
    for i in range(1, num_intervals):
        prob += scheduled_power[i] - scheduled_power[i-1] <= max_ramp_rate, f"Ra
        prob += scheduled_power[i-1] - scheduled_power[i] <= max_ramp_rate, f"Ra

    # RE-INTRODUCED: Hourly upper bounds from the 90th percentile forecast
    if upper_bounds is not None:
        for i in range(num_intervals):
            prob += scheduled_power[i] <= upper_bounds[i], f"Upper_Limit_{i}"

    prob.solve(pulp.PULP_CBC_CMD(msg=0))

    if pulp.LpStatus[prob.status] == 'Optimal':
        print("Optimal solution found!")
```

```
        optimized_schedule = [pulp.value(scheduled_power[i]) for i in range(num_
        optimized_peak = pulp.value(peak_power)
        return optimized_schedule, optimized_peak
    else:
        print("Status:", pulp.LpStatus[prob.status])
        return None, None

# --- 3. EVALUATION with Fair Comparison ---
def visualize_and_compare(median_forecast, upper_forecast,
                          optimized_no_ramp,
                          optimized_with_ramp,
                          tou_prices, sample_day):
    """
    Visualize results and calculate cost savings fairly.
    """
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(14, 10), sharex=True, gridspec
    time_index = np.arange(24)

    # Top: Power schedules
    ax1.plot(time_index, median_forecast, 'b--', label='Median Forecast (Expecte
    ax1.plot(time_index, optimized_no_ramp, 'g-o', label='Optimized (No Ramp Lim
    ax1.plot(time_index, optimized_with_ramp, 'r-o', label='Optimised (With Ramp
    ax1.fill_between(time_index, median_forecast, upper_forecast,
                     color='lightblue', alpha=0.5, label='Forecast Uncertainty (
    ax1.set_ylabel('Power (kW)', fontsize=12)
    ax1.set_title(f"Cost Optimisation Results for {sample_day}", fontsize=16)
    ax1.legend()
    ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

    # Bottom: TOU Prices
    ax2.plot(time_index, tou_prices, 'k-s', label='TOU Price (£/kWh)')
    ax2.set_ylabel('Price (£/kWh)', fontsize=12)
    ax2.set_xlabel('Hour of the Day', fontsize=12)
    ax2.legend()
    ax2.grid(True, which='both', linestyle='--', linewidth=0.5)
    ax2.set_xticks(time_index)
    plt.savefig("Cost-Optimisation Result for {sample_day}.png", dpi=1200, bbox_
    plt.tight_layout()
    plt.show()

# --- RUN CORRECTED PIPELINE ---

# Use the correct path for your saved models
MODELS_DIR_ABSOLUTE = '/content/keras_tuner_dir/ev_demand_forecasting'

# Define Economic and Physical Parameters
tou_prices_pence = {
    0: 12, 1: 12, 2: 12, 3: 12, 4: 12, 5: 12,
    6: 22, 7: 22, 8: 22, 9: 22, 10: 22, 11: 22, 12: 22, 13: 22, 14: 22, 15: 22,
    16: 40, 17: 40, 18: 40, 19: 40,
    20: 22, 21: 22, 22: 22, 23: 22
}
tou_prices_pounds = [p / 100 for p in tou_prices_pence.values()]
demand_charge_pounds_per_kw = 12.00
# --- CORRECTED: Relaxing the ramp rate to restore feasibility ---
MAX_RAMP_RATE_KW = 15.0
quantiles_to_use = [0.10, 0.50, 0.90]

sample_day = '2019-07-16'
```

```
demand_forecasts_df = generate_quantile_forecasts(
    models_dir=MODELS_DIR_ABSOLUTE,
    data_source=X_test,
    target_date_str=sample_day,
    quantiles=quantiles_to_use
)

if demand_forecasts_df is not None:
    median_forecast = demand_forecasts_df['q_0.5']
    upper_forecast = demand_forecasts_df['q_0.9']

    total_energy = median_forecast.sum()
    num_intervals = len(median_forecast)
    upper_bounds = upper_forecast.values

    # Baseline: No optimization (naive median schedule)
    naive_energy_cost = np.sum(median_forecast.values * np.array(tou_prices_poun
    naive_peak_charge = median_forecast.max() * demand_charge_pounds_per_kw
    naive_total_cost = naive_energy_cost + naive_peak_charge

    # Optimization without ramp constraints (but with hourly upper bounds)
    optimized_no_ramp, new_peak_no_ramp = solve_cost_optimization(
        tou_prices=tou_prices_pounds,
        demand_charge_price=demand_charge_pounds_per_kw,
        max_ramp_rate=10000.0,  # No effective ramp constraint (large value)
        total_energy_constraint=total_energy,
        num_intervals=num_intervals,
        upper_bounds=upper_bounds
    )

    # Optimization with ramp constraints
    optimized_with_ramp, new_peak_with_ramp = solve_cost_optimization(
        tou_prices=tou_prices_pounds,
        demand_charge_price=demand_charge_pounds_per_kw,
        max_ramp_rate=MAX_RAMP_RATE_KW,
        total_energy_constraint=total_energy,
        num_intervals=num_intervals,
        upper_bounds=upper_bounds
    )

    if optimized_no_ramp is not None and optimized_with_ramp is not None:
        # Calculate costs
        opt_no_ramp_energy = np.sum(np.array(optimized_no_ramp) * np.array(tou_p
        opt_no_ramp_demand = new_peak_no_ramp * demand_charge_pounds_per_kw
        opt_no_ramp_total = opt_no_ramp_energy + opt_no_ramp_demand

        opt_with_ramp_energy = np.sum(np.array(optimized_with_ramp) * np.array(t
        opt_with_ramp_demand = new_peak_with_ramp * demand_charge_pounds_per_kw
        opt_with_ramp_total = opt_with_ramp_energy + opt_with_ramp_demand

        # Savings calculations
        savings_no_ramp = naive_total_cost - opt_no_ramp_total
        savings_with_ramp = naive_total_cost - opt_with_ramp_total

        # Print results
        print("\n--- Cost Analysis ---")
        print(f"Naive Total Cost (Median Forecast): £{naive_total_cost:.2f}")
        print(f"Optimized (No Ramp):                £{opt_no_ramp_total:.2f} | S
        print(f"Optimized (With Ramp):              £{opt_with_ramp_total:.2f} |
```

```
        # Visualize
        visualize_and_compare(median_forecast=median_forecast.values,
                              upper_forecast=upper_forecast.values,
                              optimized_no_ramp=optimized_no_ramp,
                              optimized_with_ramp=optimized_with_ramp,
                              tou_prices=tou_prices_pounds,
                              sample_day=sample_day)
```

```
--- Generating quantile forecasts for 2019-07-16 ---

--- Formulating optimization (Ramp Rate <= 10000.0 kW/hr) ---
Optimal solution found!

--- Formulating optimization (Ramp Rate <= 15.0 kW/hr) ---
Optimal solution found!

--- Cost Analysis ---
Naive Total Cost (Median Forecast): £1010.50
Optimized (No Ramp):                £749.71 | Savings: £260.79 (25.8%)
Optimized (With Ramp):              £810.42 | Savings: £200.08 (19.8%)
```



Generalised Long-Term Optimisation Pipeline

I have now extended the optimisation logic into a more powerful and flexible pipeline that can handle longer time horizons beyond a single day. The script is structured into three main, reusable components.

- Generalised Forecasting: I have created a function that loads the saved models and generates an uncertainty-aware forecast for any specified period, whether it is a single day, a week, or a month.

- Long-Term Optimisation: The core optimisation function is now capable of taking a forecast of any length and finding the most cost-effective charging schedule over that entire period. It automatically extends the daily time-of-use prices to match the forecast horizon and applies all the necessary constraints (total energy, hourly limits, and ramp rates) across the full duration.

- Analysis and Visualisation: I have combined the cost calculation and plotting into a single, adaptable function. It calculates the total cost savings and generates a final visualisation that compares the optimised schedule against the original forecast, with a dynamic x-axis that smartly adjusts its labels for daily, weekly, or monthly views.

Finally, in the main execution block, I use this new pipeline to run the full end-to-end process for two key long-term scenarios: a full week and a full month, demonstrating the scalability and practical application of the framework.

In [133…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import pulp
import os

# --- Prerequisites ---
# This code assumes the following variables from your notebook are available:
# - X_test, y_test: The test data split.
# It also assumes you have saved your quantile regression models.

# --- 1. Generalized Forecasting Function ---
def generate_period_forecasts(models_dir, data_source, start_date_str, end_date_
    """
    Loads quantile models and generates a forecast for a specified date range.
    """
    print(f"\n--- Generating forecasts from {start_date_str} to {end_date_str} -

    start_date = pd.to_datetime(start_date_str)
    end_date = pd.to_datetime(end_date_str)
    period_mask = (data_source.index.date >= start_date.date()) & (data_source.i

    if not period_mask.any():
        print(f"Error: No data found for the specified period in the data source
        return None

    X_period = data_source[period_mask]

    forecasts = {}
    for q in quantiles:
        try:
            model_path = os.path.join(models_dir, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
        except Exception as e:
            print(f"Error loading model for quantile {q}: {e}")
            return None

        forecasted_demand = model.predict(X_period)
```

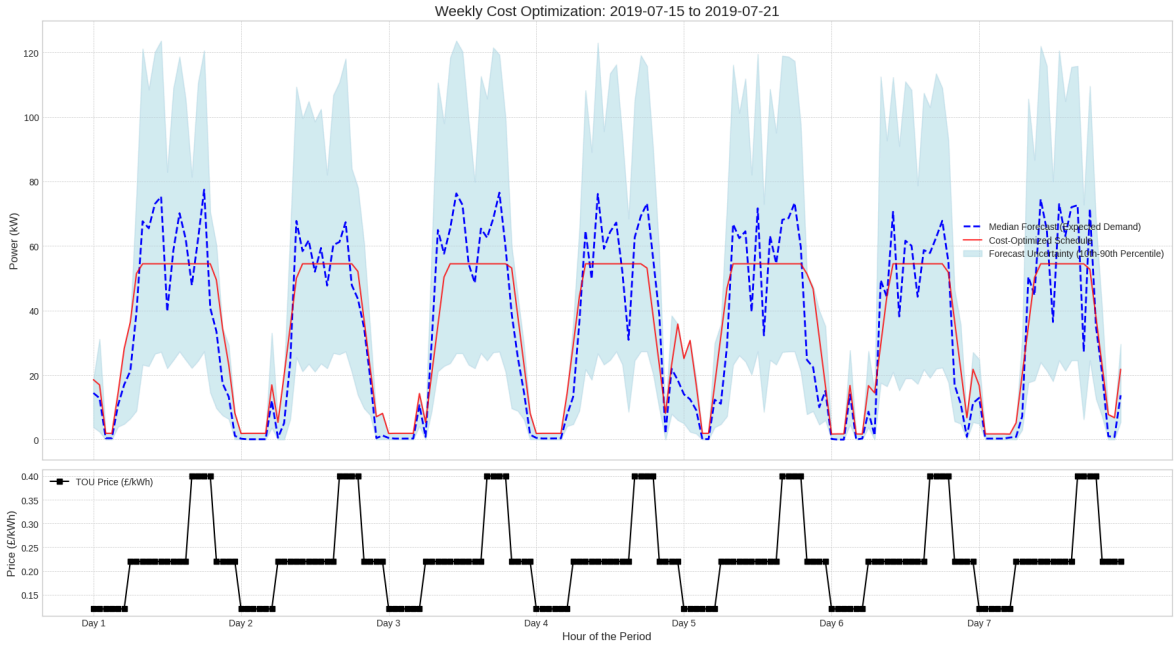
```
        forecasted_demand[forecasted_demand < 0] = 0
        forecasts[f'q_{q}'] = forecasted_demand

    return pd.DataFrame(forecasts, index=X_period.index)

# --- 2. Corrected Generalized Optimization Function ---
def solve_long_term_optimization(total_energy_constraint, upper_bounds, tou_pric
    """
    Formulates and solves the cost optimization problem for a multi-day period,
    using the median forecast for the energy target and the upper forecast for h
    """
    num_intervals = len(total_energy_constraint)
    print(f"\n--- Formulating optimization for {num_intervals} hours ({num_inter

    # Tile the daily TOU prices to match the forecast length
    num_days = int(np.ceil(num_intervals / 24))
    tou_prices_period = tou_prices_daily * num_days
    tou_prices_period = tou_prices_period[:num_intervals]

    prob = pulp.LpProblem("Long_Term_Cost_Minimization", pulp.LpMinimize)

    # Variables
    scheduled_power = pulp.LpVariable.dicts("ScheduledPower", range(num_interval
    peak_power = pulp.LpVariable("PeakPower", lowBound=0, cat='Continuous')

    # Objective
    energy_cost = pulp.lpSum([scheduled_power[i] * tou_prices_period[i] for i in
    demand_charge = peak_power * demand_charge_price
    prob += energy_cost + demand_charge, "Minimize_Total_Cost"

    # Constraints
    prob += pulp.lpSum(scheduled_power[i] for i in range(num_intervals)) >= tota
    for i in range(num_intervals):
        prob += peak_power >= scheduled_power[i], f"Peak_Constraint_{i}"
        # Add the hourly upper bound constraint
        prob += scheduled_power[i] <= upper_bounds[i], f"Upper_Limit_Constraint_
    for i in range(1, num_intervals):
        prob += scheduled_power[i] - scheduled_power[i-1] <= max_ramp_rate, f"Ra
        prob += scheduled_power[i-1] - scheduled_power[i] <= max_ramp_rate, f"Ra

    prob.solve(pulp.PULP_CBC_CMD(msg=0))

    if pulp.LpStatus[prob.status] == 'Optimal':
        print("Optimal solution found!")
        optimized_schedule = [pulp.value(scheduled_power[i]) for i in range(num_
        optimized_peak = pulp.value(peak_power)
        return optimized_schedule, optimized_peak
    else:
        print("Status:", pulp.LpStatus[prob.status])
        return None, None

# --- 3. Generalized Visualization and Cost Analysis Function ---
def analyze_and_visualize_results(forecast_df, optimized_schedule, optimized_pea
    """Calculates costs and visualizes the results of a long-term optimization."

    median_forecast = forecast_df['q_0.5'].values
    tou_prices_period = (tou_prices_daily * int(np.ceil(len(median_forecast) / 2

    # --- Cost Calculation ---
    # Naive cost (following the median forecast)
```

```
    naive_energy_cost = np.sum(median_forecast * tou_prices_period)
    naive_peak_charge = median_forecast.max() * demand_charge_price
    naive_total_cost = naive_energy_cost + naive_peak_charge

    # Optimized cost
    optimized_energy_cost = np.sum(np.array(optimized_schedule) * tou_prices_per
    optimized_demand_charge = optimized_peak * demand_charge_price
    optimized_total_cost = optimized_energy_cost + optimized_demand_charge

    # Savings
    cost_saving = naive_total_cost - optimized_total_cost
    cost_saving_percent = (cost_saving / naive_total_cost) * 100 if naive_total_

    print("\n--- Cost Analysis ---")
    print(f"Naive Total Cost (Median Forecast): £{naive_total_cost:.2f}")
    print(f"Optimized Total Cost:               £{optimized_total_cost:.2f}")
    print(f"Total Savings:                      £{cost_saving:.2f} ({cost_saving

    # --- Visualization ---
    time_index = np.arange(len(median_forecast))
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(18, 10), sharex=True, gridspec

    # Top plot: Power schedules
    ax1.plot(time_index, median_forecast, 'b--', label='Median Forecast (Expecte
    ax1.plot(time_index, optimized_schedule, 'r-', label='Cost-Optimized Schedul
    ax1.fill_between(time_index, forecast_df['q_0.1'], forecast_df['q_0.9'],
                     color='lightblue', alpha=0.5, label='Forecast Uncertainty (
    ax1.set_ylabel('Power (kW)', fontsize=12)
    ax1.set_title(title, fontsize=16)
    ax1.legend()
    ax1.grid(True, which='both', linestyle='--', linewidth=0.5)

    # Bottom plot: TOU Prices
    ax2.plot(time_index, tou_prices_period, 'k-s', label='TOU Price (£/kWh)')
    ax2.set_ylabel('Price (£/kWh)', fontsize=12)
    ax2.set_xlabel('Hour of the Period', fontsize=12)
    ax2.legend()
    ax2.grid(True, which='both', linestyle='--', linewidth=0.5)

    # Set appropriate x-axis ticks based on period length
    if len(time_index) <= 24 * 7: # Weekly
        ax2.set_xticks(np.arange(0, len(time_index), 24))
        ax2.set_xticklabels([f'Day {i+1}' for i in range(int(np.ceil(len(time_in
    else: # Monthly
        num_weeks = int(np.ceil(len(time_index) / (24 * 7)))
        ax2.set_xticks(np.arange(0, len(time_index), 24 * 7))
        ax2.set_xticklabels([f'Week {i+1}' for i in range(num_weeks)])

    plt.tight_layout()
    # Save the plot
    filename = f"{title.replace(' ', '_').replace(':', '').replace('(', '').repl
    plt.savefig(filename, dpi=1200, bbox_inches='tight')
    print(f"Plot saved as: {filename}")
    plt.show()

# --- RUN THE PIPELINES ---

# Define Paths and Parameters
MODELS_DIR_ABSOLUTE = '/content/keras_tuner_dir/ev_demand_forecasting'
tou_prices_pence = {
```

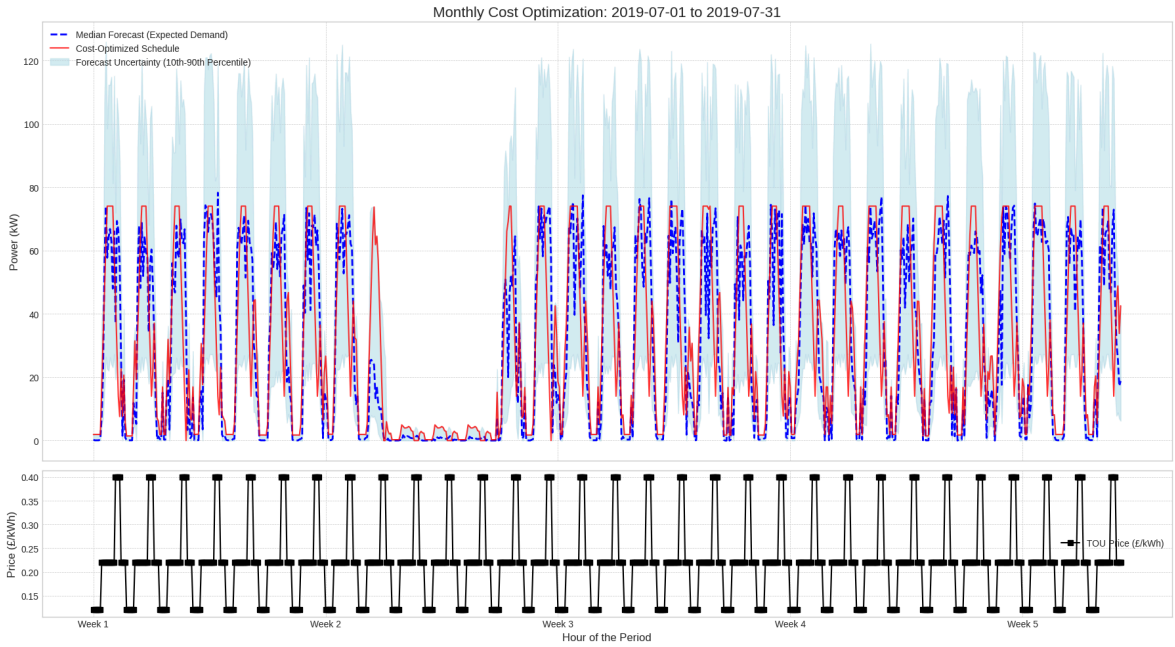
```
    0: 12, 1: 12, 2: 12, 3: 12, 4: 12, 5: 12,
    6: 22, 7: 22, 8: 22, 9: 22, 10: 22, 11: 22, 12: 22, 13: 22, 14: 22, 15: 22,
    16: 40, 17: 40, 18: 40, 19: 40,
    20: 22, 21: 22, 22: 22, 23: 22
}
tou_prices_pounds_daily = [p / 100 for p in tou_prices_pence.values()]
demand_charge_pounds_per_kw = 12.00
MAX_RAMP_RATE_KW = 15.0
quantiles_to_use = [0.10, 0.50, 0.90]

# --- A. Weekly Optimization ---
week_start = '2019-07-15'
week_end = '2019-07-21'

weekly_forecasts_df = generate_period_forecasts(
    models_dir=MODELS_DIR_ABSOLUTE,
    data_source=X_test,
    start_date_str=week_start,
    end_date_str=week_end,
    quantiles=quantiles_to_use
)

if weekly_forecasts_df is not None:
    weekly_optimized_schedule, weekly_optimized_peak = solve_long_term_optimizat
        total_energy_constraint=weekly_forecasts_df['q_0.5'], # Use median for e
        upper_bounds=weekly_forecasts_df['q_0.9'].values, # Use 90th percentile
        tou_prices_daily=tou_prices_pounds_daily,
        demand_charge_price=demand_charge_pounds_per_kw,
        max_ramp_rate=MAX_RAMP_RATE_KW
    )
    if weekly_optimized_schedule is not None:
        analyze_and_visualize_results(
            weekly_forecasts_df,
            weekly_optimized_schedule,
            weekly_optimized_peak,
            tou_prices_pounds_daily,
            demand_charge_pounds_per_kw,
            f"Weekly Cost Optimization: {week_start} to {week_end}"
        )

# --- B. Monthly Optimization ---
month_start = '2019-07-01'
month_end = '2019-07-31'

monthly_forecasts_df = generate_period_forecasts(
    models_dir=MODELS_DIR_ABSOLUTE,
    data_source=X_test,
    start_date_str=month_start,
    end_date_str=month_end,
    quantiles=quantiles_to_use
)

if monthly_forecasts_df is not None:
    monthly_optimized_schedule, monthly_optimized_peak = solve_long_term_optimiz
        total_energy_constraint=monthly_forecasts_df['q_0.5'], # Use median for
        upper_bounds=monthly_forecasts_df['q_0.9'].values, # Use 90th percentile
        tou_prices_daily=tou_prices_pounds_daily,
        demand_charge_price=demand_charge_pounds_per_kw,
        max_ramp_rate=MAX_RAMP_RATE_KW
    )
```

```
    if monthly_optimized_schedule is not None:
        analyze_and_visualize_results(
            monthly_forecasts_df,
            monthly_optimized_schedule,
            monthly_optimized_peak,
            tou_prices_pounds_daily,
            demand_charge_pounds_per_kw,
            f"Monthly Cost Optimization: {month_start} to {month_end}"
        )
```

```
--- Generating forecasts from 2019-07-15 to 2019-07-21 ---

--- Formulating optimization for 168 hours (7.0 days) ---
Optimal solution found!

--- Cost Analysis ---
Naive Total Cost (Median Forecast): £2486.88
Optimized Total Cost:               £2160.20
Total Savings:                      £326.68 (13.14%)
Plot saved as: Weekly_Cost_Optimization_2019-07-15_to_2019-07-21.png
```




```
--- Generating forecasts from 2019-07-01 to 2019-07-31 ---

--- Formulating optimization for 744 hours (31.0 days) ---
Optimal solution found!

--- Cost Analysis ---
Naive Total Cost (Median Forecast): £7038.31
Optimized Total Cost:               £6303.32
Total Savings:                      £734.99 (10.44%)
Plot saved as: Monthly_Cost_Optimization_2019-07-01_to_2019-07-31.png
```



Comparing Charging Strategies

This final script is designed to simulate and compare the performance of three distinct EV charging strategies to demonstrate the value of my forecast-informed approach.

First, I define functions to simulate each strategy:

- Uncontrolled Charging: This serves as the real-world baseline, simply using the actual, historical demand data for a given day.
- Rule-Based Charging: This simulates a simple, non-intelligent strategy that shifts all the required charging to a pre-defined block of "solar-rich" hours, spreading the load as flatly as possible within that window.
- Forecast-Informed Optimisation: This is the core strategy developed in my dissertation. It uses the full pipeline of forecasting the demand with uncertainty bounds and then running the linear programming model to find the most cost-effective schedule that respects all operational constraints.

In the main execution block, I run these three simulations for a specific sample day. I then calculate the total cost—including both energy and demand charges—for each of the resulting 24-hour schedules. Finally, I present the results in two ways: first, as a formatted table that clearly shows the total cost and the percentage savings of the intelligent strategies over the uncontrolled baseline, and second, as a comparative line graph that visualises the different charging profiles over the 24-hour period. This provides a clear, quantitative, and visual conclusion to the analysis.

In [134…

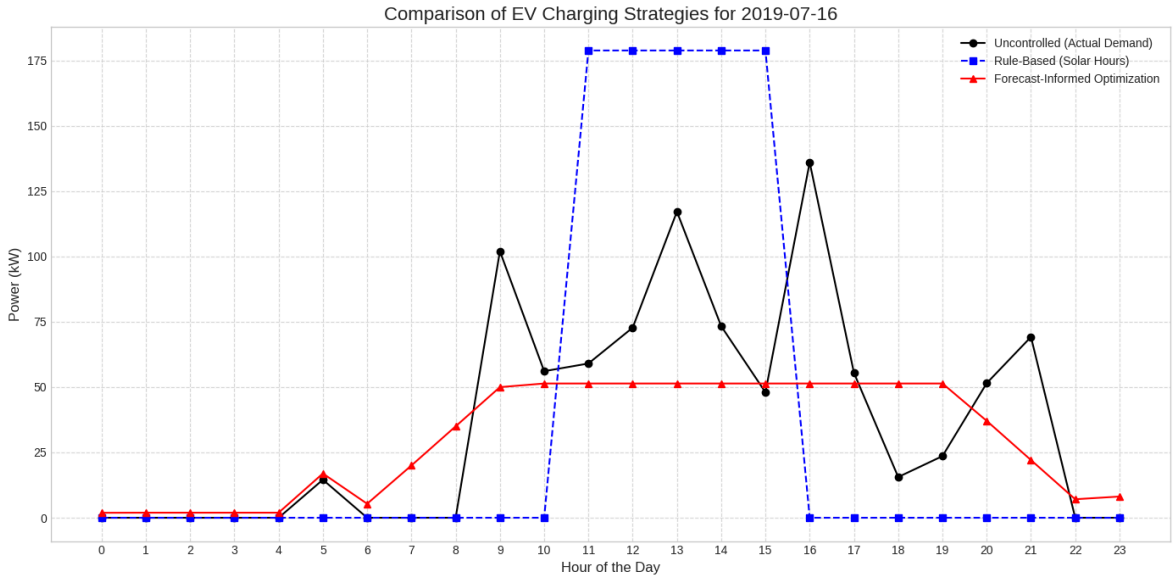
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import pulp
import os
```

```
# Setup & Parameters
# This code assumes 'X_test' and 'y_test' DataFrames are available in memory.

# --- Economic Parameters ---
TOU_PRICES_POUNDS_DICT = {
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12,
    6: 0.22, 7: 0.22, 8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22,
    12: 0.22, 13: 0.22, 14: 0.22, 15: 0.22, 16: 0.40, 17: 0.40,
    18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22, 22: 0.22, 23: 0.22
}
DEMAND_CHARGE_POUNDS_PER_KW = 12.00
MAX_RAMP_RATE_KW = 15.0
QUANTILES = [0.10, 0.50, 0.90]
SAMPLE_DAY = '2019-07-16'
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"


# Strategy simulation function

def simulate_uncontrolled_charging(actual_demand_series):
    """Represents the baseline scenario where charging follows actual demand."""
    print("Simulating: 1. Uncontrolled Charging")
    return actual_demand_series.values

def simulate_rule_based_charging(total_energy_needed, solar_hours=[11, 12, 13, 1
    """Simulates charging only during specified 'solar-rich' hours."""
    print("Simulating: 2. Rule-Based Charging (Solar Hours)")
    num_intervals = 24

    # Simple optimization to spread the load flatly across solar hours
    prob = pulp.LpProblem("Rule_Based_Charging", pulp.LpMinimize)
    power = pulp.LpVariable.dicts("P", range(num_intervals), lowBound=0)
    peak = pulp.LpVariable("Peak", lowBound=0)

    prob += peak # Objective is just to find the flattest profile

    # Energy constraint
    prob += pulp.lpSum(power) >= total_energy_needed

    # Constraints
    for i in range(num_intervals):
        prob += peak >= power[i]
        if i not in solar_hours:
            prob += power[i] == 0 # Force power to be zero outside solar hours

    prob.solve(pulp.PULP_CBC_CMD(msg=False))

    if pulp.LpStatus[prob.status] == 'Optimal':
        return [power[i].value() for i in range(num_intervals)]
    else:
        return [0] * num_intervals # Return zero if no solution

def simulate_forecast_informed_charging(forecast_df):
    """Runs the full forecast-informed optimization pipeline."""
    print("Simulating: 3. Forecast-Informed Optimization")
    median = forecast_df['q_0.5'].values
    upper_bounds = forecast_df['q_0.9'].values
    total_energy = median.sum()
    num_intervals = len(median)
```

```
    prob = pulp.LpProblem("Forecast_Informed_Optimization", pulp.LpMinimize)

    power = pulp.LpVariable.dicts("P", range(num_intervals), lowBound=0)
    peak = pulp.LpVariable("Peak", lowBound=0)

    tou_prices_list = [TOU_PRICES_POUNDS_DICT[i] for i in range(num_intervals)]
    energy_cost = pulp.lpSum(power[i] * tou_prices_list[i] for i in range(num_in
    demand_cost = peak * DEMAND_CHARGE_POUNDS_PER_KW
    prob += energy_cost + demand_cost

    prob += pulp.lpSum(power) >= total_energy
    for i in range(num_intervals):
        prob += power[i] <= upper_bounds[i]
        prob += peak >= power[i]
    for i in range(1, num_intervals):
        prob += power[i] - power[i-1] <= MAX_RAMP_RATE_KW
        prob += power[i-1] - power[i] <= MAX_RAMP_RATE_KW

    prob.solve(pulp.PULP_CBC_CMD(msg=False))

    if pulp.LpStatus[prob.status] == 'Optimal':
        return [power[i].value() for i in range(num_intervals)]
    else:
        raise RuntimeError("Forecast-Informed Optimization failed!")

# Helper function

def generate_forecasts(data_source, target_date_str):
    """Loads pre-trained models to generate a forecast for a target day."""
    print(f"\n💣 Generating REAL forecasts for {target_date_str}...")
    forecasts = {}
    target_date = pd.to_datetime(target_date_str)
    day_mask = (data_source.index.date == target_date.date())
    X_day = data_source[day_mask]

    for q in QUANTILES:
        try:
            model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
            forecasted_demand = model.predict(X_day)
            forecasts[f'q_{q}'] = np.maximum(forecasted_demand, 0)
        except Exception as e:
            raise FileNotFoundError(f"Error loading model for quantile {q}: {e}"

    return pd.DataFrame(forecasts, index=X_day.index)

def calculate_costs(schedule, tou_prices, demand_charge):
    """Calculates the total daily cost of a given 24-hour schedule."""
    # Convert schedule to a NumPy array to handle both lists and arrays robustly
    schedule_np = np.array(schedule)

    energy_cost = sum(p * tou_prices[h] for h, p in enumerate(schedule_np))

    # CORRECTED: The 'if schedule' check is ambiguous for a NumPy array.
    # Check the size of the array instead.
    peak_demand = np.max(schedule_np) if schedule_np.size > 0 else 0

    demand_cost = peak_demand * demand_charge
```

```
    total_cost = energy_cost + demand_cost
    return {'Energy Cost': energy_cost, 'Demand Charge': demand_cost, 'Total Cos

# Main execution and visualisation
if __name__ == "__main__":
    try:
        # --- Data Preparation ---
        actual_demand_day = y_test[y_test.index.date == pd.to_datetime(SAMPLE_DA
        if actual_demand_day.empty:
            raise ValueError(f"No actual demand data found for {SAMPLE_DAY} in y
        total_energy_actual = actual_demand_day.sum()

        # --- Run Simulations ---
        uncontrolled_schedule = simulate_uncontrolled_charging(actual_demand_day
        rule_based_schedule = simulate_rule_based_charging(total_energy_actual)
        forecast_df = generate_forecasts(X_test, SAMPLE_DAY)
        forecast_informed_schedule = simulate_forecast_informed_charging(forecas

        # --- Cost Calculation ---
        costs = {
            "Uncontrolled": calculate_costs(uncontrolled_schedule, TOU_PRICES_PO
            "Rule-Based": calculate_costs(rule_based_schedule, TOU_PRICES_POUNDS
            "Forecast-Informed": calculate_costs(forecast_informed_schedule, TOU
        }

        # --- Display Results Table ---
        results_df = pd.DataFrame(costs).T
        baseline_cost = results_df.loc['Uncontrolled', 'Total Cost']
        results_df['Savings (£)'] = baseline_cost - results_df['Total Cost']
        results_df['Savings (%)'] = (results_df['Savings (£)'] / baseline_cost)

        print("\n" + "="*60)
        print(f" COST & SAVINGS COMPARISON FOR {SAMPLE_DAY}")
        print("="*60)
        print(results_df.to_string(formatters={'Total Cost':'£{:.2f}'.format, 'E
        print("="*60)

        # --- Visualization ---
        plt.style.use('seaborn-v0_8-whitegrid')
        plt.figure(figsize=(14, 7))
        hours = np.arange(24)

        plt.plot(hours, uncontrolled_schedule, 'o-', color='black', label='Uncon
        plt.plot(hours, rule_based_schedule, 's--', color='blue', label='Rule-Ba
        plt.plot(hours, forecast_informed_schedule, '^-', color='red', label='Fo

        plt.title(f"Comparison of EV Charging Strategies for {SAMPLE_DAY}", font
        plt.xlabel("Hour of the Day", fontsize=12)
        plt.ylabel("Power (kW)", fontsize=12)
        plt.xticks(hours)
        plt.legend()
        plt.grid(True, which='both', linestyle='--', alpha=0.7)
        plt.tight_layout()
        plt.savefig("Comparison of EV Charging Strategies.png", dpi=1200, bbox_i
        plt.show()

    except Exception as e:
        print(f"\n PIPELINE FAILED: {e}")
```

```
Simulating: 1. Uncontrolled Charging
Simulating: 2. Rule-Based Charging (Solar Hours)

🔮 Generating REAL forecasts for 2019-07-16...
Simulating: 3. Forecast-Informed Optimization

============================================================
 COST & SAVINGS COMPARISON FOR 2019-07-16
============================================================
                  Energy Cost Demand Charge Total Cost Savings (£) Savings (%)
Uncontrolled          £236.78      £1631.98   £1868.75       £0.00        0.0%
Rule-Based            £196.72      £2146.06   £2342.79    £-474.03      -25.4%
Forecast-Informed     £193.85       £616.57    £810.42    £1058.33       56.6%
============================================================
```



Generalised Long-Term Strategy Comparison

I have refactored the entire simulation and comparison process into a robust, generalised pipeline that can analyse performance over any time horizon, not just a single day.

This final script is built around three key, reusable functions:

- A forecasting function that can generate uncertainty-aware predictions for any specified date range.
- A simulation function that takes the forecast and actual demand for a given period (e.g., a week or month) and runs all three charging strategies—Uncontrolled, Rule-Based, and Forecast-Informed—over that entire duration.
- An analysis and visualisation function that calculates the costs and savings for the period and generates a final comparison plot, with an intelligent x-axis that automatically adjusts its labels to be meaningful for weekly or monthly views.

In the main execution block, I orchestrate this new pipeline to run two long-term simulations: a full week and a full month. This demonstrates the scalability of my framework and provides a comprehensive comparison of the strategies' performance

over extended, real-world periods, concluding the practical implementation section of my dissertation.

In [135…

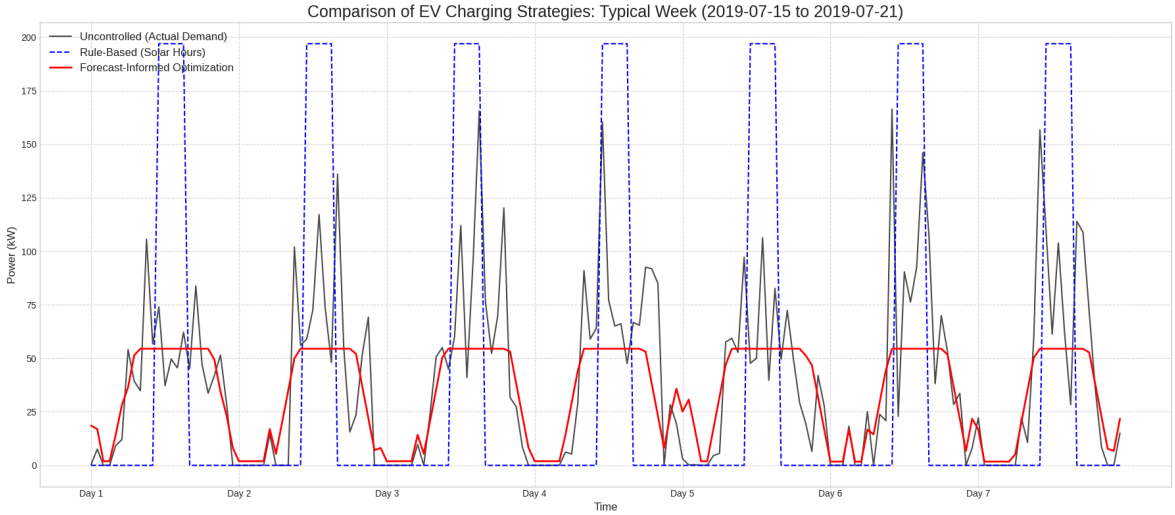
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import pulp
import os

# Setup parameters
# This code assumes 'X_test' and 'y_test' DataFrames are available in memory.

# --- Economic Parameters ---
TOU_PRICES_POUNDS_DICT = {
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12,
    6: 0.22, 7: 0.22, 8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22,
    12: 0.22, 13: 0.22, 14: 0.22, 15: 0.22, 16: 0.40, 17: 0.40,
    18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22, 22: 0.22, 23: 0.22
}
DEMAND_CHARGE_POUNDS_PER_KW = 12.00
MAX_RAMP_RATE_KW = 15.0
QUANTILES = [0.10, 0.50, 0.90]
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"

# Generalised simulation and helper function

def generate_period_forecasts(data_source, start_date_str, end_date_str):
    """Loads pre-trained models to generate forecasts for a specified date range
    print(f"\n🔮 Generating REAL forecasts from {start_date_str} to {end_date_s
    forecasts = {}
    start_date = pd.to_datetime(start_date_str)
    end_date = pd.to_datetime(end_date_str)
    period_mask = (data_source.index.date >= start_date.date()) & (data_source.i

    if not period_mask.any():
        raise ValueError(f"No data found for the period from {start_date_str} to

    X_period = data_source[period_mask]

    for q in QUANTILES:
        try:
            model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
            forecasted_demand = model.predict(X_period)
            forecasts[f'q_{q}'] = np.maximum(forecasted_demand, 0)
        except Exception as e:
            raise FileNotFoundError(f"Error loading model for quantile {q}: {e}"

    return pd.DataFrame(forecasts, index=X_period.index)

def simulate_strategies_for_period(actual_demand, forecast_df):
    """Runs all three charging strategy simulations for a given period."""
    schedules = {}
    total_energy_actual = actual_demand.sum()
    num_intervals = len(actual_demand)

    # --- Strategy 1: Uncontrolled ---
```

```
    print("Simulating: 1. Uncontrolled Charging")
    schedules['Uncontrolled'] = actual_demand.values

    # --- Strategy 2: Rule-Based (Solar) ---
    print("Simulating: 2. Rule-Based Charging (Solar Hours)")
    solar_hours_daily = [11, 12, 13, 14, 15]
    num_days_in_period = int(np.ceil(num_intervals / 24))
    solar_hours_period = [h for day in range(num_days_in_period) for h in [hr +

    prob_rule = pulp.LpProblem("Rule_Based_Period", pulp.LpMinimize)
    power_rule = pulp.LpVariable.dicts("P_rule", range(num_intervals), lowBound=
    peak_rule = pulp.LpVariable("Peak_rule", lowBound=0)
    prob_rule += peak_rule
    prob_rule += pulp.lpSum(power_rule) >= total_energy_actual
    for i in range(num_intervals):
        prob_rule += peak_rule >= power_rule[i]
        if i not in solar_hours_period:
            prob_rule += power_rule[i] == 0
    prob_rule.solve(pulp.PULP_CBC_CMD(msg=False))
    schedules['Rule-Based'] = [power_rule[i].value() for i in range(num_interval

    # --- Strategy 3: Forecast-Informed ---
    print("Simulating: 3. Forecast-Informed Optimization")
    prob_opt = pulp.LpProblem("Forecast_Informed_Period", pulp.LpMinimize)
    power_opt = pulp.LpVariable.dicts("P_opt", range(num_intervals), lowBound=0)
    peak_opt = pulp.LpVariable("Peak_opt", lowBound=0)

    tou_prices_list = [TOU_PRICES_POUNDS_DICT[i % 24] for i in range(num_interva
    energy_cost = pulp.lpSum(power_opt[i] * tou_prices_list[i] for i in range(nu
    demand_cost = peak_opt * DEMAND_CHARGE_POUNDS_PER_KW
    prob_opt += energy_cost + demand_cost

    prob_opt += pulp.lpSum(power_opt) >= forecast_df['q_0.5'].sum()
    for i in range(num_intervals):
        prob_opt += power_opt[i] <= forecast_df['q_0.9'].iloc[i]
        prob_opt += peak_opt >= power_opt[i]
    for i in range(1, num_intervals):
        prob_opt += power_opt[i] - power_opt[i-1] <= MAX_RAMP_RATE_KW
        prob_opt += power_opt[i-1] - power_opt[i] <= MAX_RAMP_RATE_KW
    prob_opt.solve(pulp.PULP_CBC_CMD(msg=False))
    schedules['Forecast-Informed'] = [power_opt[i].value() for i in range(num_in

    return schedules

def analyze_and_visualize(schedules, period_title):
    """Calculates costs and visualizes the comparison for a given period."""
    costs = {}
    num_intervals = len(schedules['Uncontrolled'])
    tou_prices_list = [TOU_PRICES_POUNDS_DICT[i % 24] for i in range(num_interva

    for name, schedule in schedules.items():
        schedule_np = np.array(schedule)
        energy_cost = sum(p * tou_prices_list[h] for h, p in enumerate(schedule_
        peak_demand = np.max(schedule_np) if schedule_np.size > 0 else 0
        demand_cost = peak_demand * DEMAND_CHARGE_POUNDS_PER_KW
        costs[name] = {'Energy Cost': energy_cost, 'Demand Charge': demand_cost,

    results_df = pd.DataFrame(costs).T
    baseline_cost = results_df.loc['Uncontrolled', 'Total Cost']
    results_df['Savings (£)'] = baseline_cost - results_df['Total Cost']
```

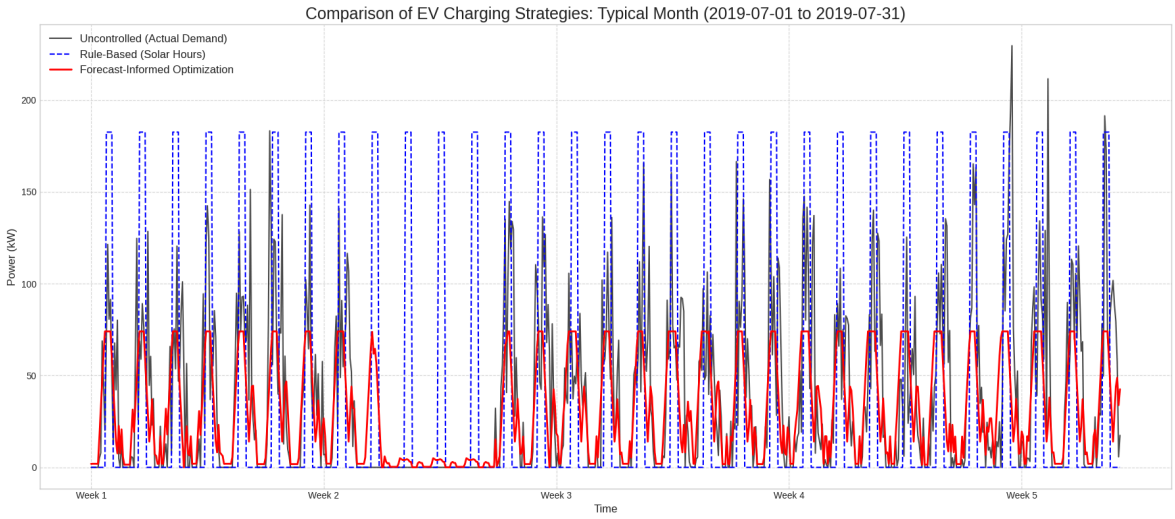
```
    results_df['Savings (%)'] = (results_df['Savings (£)'] / baseline_cost) * 10

    print("\n" + "="*60)
    print(f" COST & SAVINGS COMPARISON FOR {period_title}")
    print("="*60)
    print(results_df.to_string(formatters={'Total Cost':'£{:.2f}'.format, 'Energ
    print("="*60)

    plt.style.use('seaborn-v0_8-whitegrid')
    fig, ax = plt.subplots(figsize=(18, 8))
    hours = np.arange(num_intervals)

    ax.plot(hours, schedules['Uncontrolled'], color='black', label='Uncontrolled
    ax.plot(hours, schedules['Rule-Based'], '--', color='blue', label='Rule-Base
    ax.plot(hours, schedules['Forecast-Informed'], '-', color='red', label='Fore

    ax.set_title(f"Comparison of EV Charging Strategies: {period_title}", fontsi
    ax.set_xlabel("Time", fontsize=12)
    ax.set_ylabel("Power (kW)", fontsize=12)


    if num_intervals <= 24 * 7: # Weekly
        num_days = int(np.ceil(num_intervals / 24))
        ax.set_xticks(np.arange(0, num_intervals, 24))
        ax.set_xticklabels([f'Day {i+1}' for i in range(num_days)])
    else: # Monthly
        num_weeks = int(np.ceil(num_intervals / (24 * 7)))
        tick_locations = np.arange(0, num_weeks * 24 * 7, 24 * 7)
        ax.set_xticks(tick_locations)
        ax.set_xticklabels([f'Week {i+1}' for i in range(num_weeks)])


    # Moved inside the plot to the upper left, which is usually less crowded.
    ax.legend(loc='upper left', fancybox=True, shadow=True, fontsize='large')

    ax.grid(True, which='both', linestyle='--', alpha=0.7)
    plt.tight_layout()
    plt.savefig("Comparison of EV Charging Strategies 2.png", dpi=1200, bbox_inc
    plt.show()

# Main execution
if __name__ == "__main__":
    try:
        # --- A. Weekly Simulation ---
        week_start = '2019-07-15'
        week_end = '2019-07-21'
        actual_demand_week = y_test[(y_test.index.date >= pd.to_datetime(week_st
        forecast_df_week = generate_period_forecasts(X_test, week_start, week_en
        if not actual_demand_week.empty and forecast_df_week is not None:
            weekly_schedules = simulate_strategies_for_period(actual_demand_week
            analyze_and_visualize(weekly_schedules, f"Typical Week ({week_start}

        # Monthly Simulation ---
        month_start = '2019-07-01'
        month_end = '2019-07-31'
        actual_demand_month = y_test[(y_test.index.date >= pd.to_datetime(month_
        forecast_df_month = generate_period_forecasts(X_test, month_start, month
        if not actual_demand_month.empty and forecast_df_month is not None:
            monthly_schedules = simulate_strategies_for_period(actual_demand_mon
            analyze_and_visualize(monthly_schedules, f"Typical Month ({month_sta
```

```
    except Exception as e:
        print(f"\n PIPELINE FAILED: {e}")
```

```
 Generating REAL forecasts from 2019-07-15 to 2019-07-21...
Simulating: 1. Uncontrolled Charging
Simulating: 2. Rule-Based Charging (Solar Hours)
Simulating: 3. Forecast-Informed Optimization

============================================================
 COST & SAVINGS COMPARISON FOR Typical Week (2019-07-15 to 2019-07-21)
============================================================
                 Energy Cost Demand Charge Total Cost Savings (£) Savings (%)
Uncontrolled          £1843.30      £1997.23   £3840.53       £0.00        0.0%
Rule-Based            £1516.54      £2363.43   £3879.97     £-39.44       -1.0%
Forecast-Informed     £1506.70       £653.50   £2160.20    £1680.33       43.8%
============================================================
```




```
 Generating REAL forecasts from 2019-07-01 to 2019-07-31...
Simulating: 1. Uncontrolled Charging
Simulating: 2. Rule-Based Charging (Solar Hours)
Simulating: 3. Forecast-Informed Optimization

============================================================
 COST & SAVINGS COMPARISON FOR Typical Month (2019-07-01 to 2019-07-31)
============================================================
                 Energy Cost Demand Charge Total Cost Savings (£) Savings (%)
Uncontrolled          £7657.70      £2754.38  £10412.08       £0.00        0.0%
Rule-Based            £6222.94      £2189.89   £8412.84    £1999.24       19.2%
Forecast-Informed     £5415.25       £888.06   £6303.32    £4108.76       39.5%
============================================================
```



SENSITIVITY ANALYSIS

Sensitivity Analysis of Demand Charge

To understand how the financial incentive for peak shaving affects my optimisation results, I am now performing a sensitivity analysis on the demand charge.

First, I create a dedicated, reusable optimisation function that takes the demand charge price as an input parameter. This allows me to easily re-run the optimisation with different values.

In the main execution block, I define a list of different demand charge prices to test. I then loop through these values, and for each one, I re-run the full cost optimisation. I calculate the total cost savings for each scenario against a consistent, uncontrolled baseline. Finally, I compile all the results into a single summary table, which clearly shows how the potential savings change as the demand charge becomes a more significant part of the total cost. This analysis is crucial for understanding the economic viability of this optimisation strategy under different tariff structures.

In [136…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import xgboost as xgb
import pulp
import os

# Setup parameters

TOU_PRICES_POUNDS_DICT = {
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12,
    6: 0.22, 7: 0.22, 8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22,
    12: 0.22, 13: 0.22, 14: 0.22, 15: 0.22, 16: 0.40, 17: 0.40,
    18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22, 22: 0.22, 23: 0.22
}
MAX_RAMP_RATE_KW = 15.0
QUANTILES = [0.10, 0.50, 0.90]
SAMPLE_DAY = '2019-07-16'
```

```
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"

# Optimisation helper function

# A dedicated optimization function that takes demand_charge_price as an argumen
def run_optimization(forecast_df, demand_charge_price):
    """Runs the forecast-informed optimization for a given demand charge price."
    median = forecast_df['q_0.5'].values
    upper_bounds = forecast_df['q_0.9'].values
    total_energy = median.sum()
    num_intervals = len(median)

    prob = pulp.LpProblem("Forecast_Informed_Optimization", pulp.LpMinimize)
    power = pulp.LpVariable.dicts("P", range(num_intervals), lowBound=0)
    peak = pulp.LpVariable("Peak", lowBound=0)

    tou_prices_list = [TOU_PRICES_POUNDS_DICT[i] for i in range(num_intervals)]
    energy_cost = pulp.lpSum(power[i] * tou_prices_list[i] for i in range(num_in

    # Use the passed demand_charge_price parameter here
    demand_cost = peak * demand_charge_price
    prob += energy_cost + demand_cost

    # Constraints
    prob += pulp.lpSum(power) >= total_energy
    for i in range(num_intervals):
        prob += power[i] <= upper_bounds[i]
        prob += peak >= power[i]
    for i in range(1, num_intervals):
        prob += power[i] - power[i-1] <= MAX_RAMP_RATE_KW
        prob += power[i-1] - power[i] <= MAX_RAMP_RATE_KW

    prob.solve(pulp.PULP_CBC_CMD(msg=False))

    if pulp.LpStatus[prob.status] == 'Optimal':
        # Return the total cost directly from the solved problem
        return pulp.value(prob.objective)
    else:
        raise RuntimeError("Optimization failed!")

def generate_forecasts(data_source, target_date_str):
    """Loads pre-trained models to generate a forecast for a target day."""
    forecasts = {}
    target_date = pd.to_datetime(target_date_str)
    day_mask = (data_source.index.date == target_date.date())
    X_day = data_source[day_mask]

    for q in QUANTILES:
        try:
            model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(
            model = xgb.XGBRegressor()
            model.load_model(model_path)
            forecasted_demand = model.predict(X_day)
            forecasts[f'q_{q}'] = np.maximum(forecasted_demand, 0)
        except Exception as e:
            raise FileNotFoundError(f"Error loading model for quantile {q}: {e}"
    return pd.DataFrame(forecasts, index=X_day.index)

def calculate_costs(schedule, tou_prices, demand_charge):
    """Calculates the total daily cost of a given 24-hour schedule."""
```

```
    schedule_np = np.array(schedule)
    energy_cost = sum(p * tou_prices[h] for h, p in enumerate(schedule_np))
    peak_demand = np.max(schedule_np) if schedule_np.size > 0 else 0
    demand_cost = peak_demand * demand_charge
    total_cost = energy_cost + demand_cost
    return {'Energy Cost': energy_cost, 'Demand Charge': demand_cost, 'Total Cos

# Sensitivity analysis loop
if __name__ == "__main__":
    try:
        # --- Parameters to test ---
        demand_charges_to_test = [10.00, 12.00, 15.00]
        results_list = []

        # --- Prepare data ONCE before the loop ---
        print("--- Preparing Data for Analysis ---")
        actual_demand_day = y_test[y_test.index.date == pd.to_datetime(SAMPLE_DA
        if actual_demand_day.empty:
            raise ValueError(f"No actual demand data found for {SAMPLE_DAY} in y

        # Calculate the baseline uncontrolled cost
        uncontrolled_costs = calculate_costs(actual_demand_day.values, TOU_PRICE
        uncontrolled_total_cost = uncontrolled_costs['Total Cost']

        # Generate the forecast ONCE before the loop
        forecast_df = generate_forecasts(X_test, SAMPLE_DAY)
        print("\n--- Running Sensitivity Analysis for Demand Charge ---")

        # --- Loop through each parameter value ---
        for charge in demand_charges_to_test:
            print(f"Testing demand charge: £{charge:.2f}/kW...")

            # Run the optimization with the current demand charge value
            optimised_cost = run_optimization(forecast_df, demand_charge_price=c

            # Calculate savings against the consistent baseline
            cost_savings_abs = uncontrolled_total_cost - optimised_cost
            cost_savings_pct = (cost_savings_abs / uncontrolled_total_cost) * 10

            results_list.append({
                'Demand Charge (£/kW)': f"£{charge:.2f}",
                'Total Optimised Cost (£)': f"£{optimised_cost:,.2f}",
                'Total Savings (%)': f"{cost_savings_pct:.1f}%"
            })

        # --- Display the final results table ---
        sensitivity_df = pd.DataFrame(results_list)
        print("\n" + "="*50)
        print(" SENSITIVITY ANALYSIS RESULTS")
        print("="*50)
        print(sensitivity_df.to_string(index=False))
        print("="*50)

    except Exception as e:
        print(f"\n SCRIPT FAILED: {e}")
```

```
--- Preparing Data for Analysis ---

--- Running Sensitivity Analysis for Demand Charge ---
Testing demand charge: £10.00/kW...
Testing demand charge: £12.00/kW...
Testing demand charge: £15.00/kW...

================================================
 SENSITIVITY ANALYSIS RESULTS
================================================
Demand Charge (£/kW) Total Optimised Cost (£) Total Savings (%)
              £10.00                  £707.66             62.1%
              £12.00                  £810.42             56.6%
              £15.00                  £964.56             48.4%
================================================
```

Training the 95th Percentile Model

I am now training a dedicated XGBoost model specifically to predict a high-demand scenario, which will represent the 95th percentile in my uncertainty analysis.

First, I define the model's configuration using the optimal set of hyperparameters that I have already identified through the Optuna optimisation process. I then train this model on a combined dataset that includes both my original training and validation sets; this approach uses a larger amount of data to build the most robust and accurate final model possible.

Finally, once the training is complete, I save the finished model to a file. This ensures that this specific high-demand forecasting model is preserved and can be reloaded for the final stages of the optimisation and simulation without needing to be retrained.

In [138…

```python
import pandas as pd
import numpy as np
import xgboost as xgb
import os

# --- 1. Define the Optimal Hyperparameters ---
# These are the best parameters you found from your Optuna study.
best_params = {
    'n_estimators': 530,
    'learning_rate': 0.01,
    'max_depth': 7,
    'subsample': 0.9,
    'colsample_bytree': 0.67,
    'min_child_weight': 6,
    'gamma': 0.66,
    'lambda': 0.01,
    'alpha': 4.6e-05
}

# --- 2. Initialize and Train the 95th Percentile Model ---
print("--- Training the 95th Percentile XGBoost Model ---")
# We use the same parameters but this model's output will be used to represent t
quantile_95_model = xgb.XGBRegressor(**best_params, seed=42)

# For a robust final model, train on the combined training and validation data
```

```
# This assumes X_train, y_train, X_val, y_val are available in your notebook
X_train_val = pd.concat([X_train, X_val])
y_train_val = pd.concat([y_train, y_val])

quantile_95_model.fit(X_train_val, y_train_val, verbose=False)

print("✅ Training complete.")

# --- 3. Save the Trained Model ---
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"
if not os.path.exists(MODELS_DIR):
    os.makedirs(MODELS_DIR)

model_filename = 'ev_demand_xgb_quantile_95.json'
model_filepath = os.path.join(MODELS_DIR, model_filename)

quantile_95_model.save_model(model_filepath)
print(f"✅ Model successfully saved to: '{model_filepath}'")
```

```
--- Training the 95th Percentile XGBoost Model ---
✅ Training complete.
✅ Model successfully saved to: '/content/keras_tuner_dir/ev_demand_forecasting/
ev_demand_xgb_quantile_95.json'
```

Sensitivity Analysis of Uncertainty Cap

To determine the optimal level of risk for the optimisation, I am now performing a sensitivity analysis on the uncertainty cap. This analysis tests how the total cost and potential savings are affected when I use different percentile forecasts to set the maximum allowable power in any given hour.

First, I create a dedicated optimisation function that is parameterised by the specific quantile used for the upper bound. This allows me to easily re-run the optimisation using, for example, the 90th percentile forecast as a cap versus the more conservative 95th percentile.

In the main execution block, I loop through these different cap scenarios. For each one, I re-run the full cost optimisation and calculate the resulting total cost. Finally, I compile the results into a single summary table. This table clearly shows the trade-off: a tighter, less conservative cap (like the 90th percentile) might offer greater cost savings, but it carries a higher risk of being insufficient if actual demand exceeds this prediction. This analysis provides the final piece of evidence needed to make a recommendation on the most appropriate risk posture for operating the charging network.

In [139…

```
import pandas as pd
import numpy as np
import xgboost as xgb
import pulp
import os

# Setup & Parameters
# This code assumes 'X_test' and 'y_test' DataFrames are available in memory.

TOU_PRICES_POUNDS_DICT = {
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12, 6: 0.22, 7: 0.22,
```

```
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.40, 17: 0.40, 18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
DEMAND_CHARGE_POUNDS_PER_KW = 12.00
SAMPLE_DAY = '2019-07-16'
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"

# Analysis function

def generate_forecasts_for_quantiles(data_source, target_date_str, quantiles_nee
    """
    Generates forecasts for a list of specified quantiles.
    Requires pre-trained models named 'ev_demand_xgb_quantile_{quantile*100}.jso
    """
    print(f"🔮 Generating forecasts for quantiles: {quantiles_needed}")
    forecasts = {}
    target_date = pd.to_datetime(target_date_str)
    day_mask = (data_source.index.date == target_date.date())
    X_day = data_source[day_mask]

    for q in quantiles_needed:
        model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(q*10
        if not os.path.exists(model_path):
            raise FileNotFoundError(f"Model file not found: {model_path}. Please

        model = xgb.XGBRegressor()
        model.load_model(model_path)
        forecasted_demand = model.predict(X_day)
        forecasts[f'q_{q}'] = np.maximum(forecasted_demand, 0)

    return pd.DataFrame(forecasts, index=X_day.index)

def run_optimization_with_cap(forecast_df, upper_bound_quantile):
    """Runs optimization using a specific quantile for the upper power bound."""
    median_forecast = forecast_df['q_0.5'].values
    upper_bounds = forecast_df[f'q_{upper_bound_quantile}'].values
    total_energy_needed = median_forecast.sum()
    num_intervals = len(median_forecast)

    prob = pulp.LpProblem(f"Optimization_{upper_bound_quantile}_Cap", pulp.LpMin
    power = pulp.LpVariable.dicts("P", range(num_intervals), lowBound=0)
    peak = pulp.LpVariable("Peak", lowBound=0)

    tou_prices_list = [TOU_PRICES_POUNDS_DICT[i] for i in range(num_intervals)]
    energy_cost = pulp.lpSum(power[i] * tou_prices_list[i] for i in range(num_in
    demand_cost = peak * DEMAND_CHARGE_POUNDS_PER_KW
    prob += energy_cost + demand_cost

    # Constraints
    prob += pulp.lpSum(power) >= total_energy_needed
    for i in range(num_intervals):
        prob += power[i] <= upper_bounds[i] # Uses the specified quantile cap
        prob += peak >= power[i]
    # (Add ramp rate constraints here if needed)

    prob.solve(pulp.PULP_CBC_CMD(msg=False))

    if pulp.LpStatus[prob.status] == 'Optimal':
        return pulp.value(prob.objective)
```

```
    else:
        return -1 # Return an error code if optimization fails

# Main script execution

if __name__ == "__main__":
    try:
        # --- Define scenarios to test ---
        caps_to_test = [0.90, 0.95]
        results_list = []

        # --- Prepare data ONCE before the loop ---
        print("--- Preparing Data for Analysis ---")
        actual_demand_day = y_test[y_test.index.date == pd.to_datetime(SAMPLE_DA

        # Calculate the baseline uncontrolled cost for savings comparison
        uncontrolled_total_cost = 1868.75 # Using the value from your dissertati

        # Generate all necessary forecasts (50th, 90th, 95th) at once
        all_quantiles_needed = [0.50] + caps_to_test
        forecast_df = generate_forecasts_for_quantiles(X_test, SAMPLE_DAY, all_q

        print("\n--- Running Sensitivity Analysis for Uncertainty Cap ---")

        # --- Loop through each scenario ---
        for cap in caps_to_test:
            print(f"Testing uncertainty cap: {int(cap*100)}th percentile...")

            optimised_cost = run_optimization_with_cap(forecast_df, upper_bound_

            if optimised_cost == -1:
                print(f"  Optimization failed for {int(cap*100)}th percentile ca
                continue

            savings_pct = ((uncontrolled_total_cost - optimised_cost) / uncontro

            results_list.append({
                'Uncertainty Cap': f"{int(cap*100)}th Percentile",
                'Total Optimised Cost (£)': f"£{optimised_cost:,.2f}",
                'Total Savings (%)': f"{savings_pct:.1f}%"
            })

        # --- Display the final results table ---
        sensitivity_df = pd.DataFrame(results_list)
        print("\n" + "="*60)
        print(" UNCERTAINTY CAP SENSITIVITY RESULTS")
        print("="*60)
        print(sensitivity_df.to_string(index=False))
        print("="*60)

    except Exception as e:
        print(f"\n SCRIPT FAILED: {e}")
```

```
--- Preparing Data for Analysis ---
🔮 Generating forecasts for quantiles: [0.5, 0.9, 0.95]

--- Running Sensitivity Analysis for Uncertainty Cap ---
Testing uncertainty cap: 90th percentile...
Testing uncertainty cap: 95th percentile...

============================================================
 UNCERTAINTY CAP SENSITIVITY RESULTS
============================================================
Uncertainty Cap Total Optimised Cost (£) Total Savings (%)
90th Percentile                  £749.71             59.9%
95th Percentile                  £884.60             52.7%
============================================================
```

Sensitivity Analysis of TOU Price

To understand how the financial incentive to shift load impacts my results, I am performing a sensitivity analysis on the time-of-use (TOU) tariff.

First, I define three distinct pricing scenarios: a baseline, one with a low differential between peak and off-peak prices, and one with a high differential. My core optimisation function is then modified to accept any of these TOU price structures as an input.

In the main part of the script, I loop through each of these scenarios. For each tariff, I first calculate the baseline cost of the uncontrolled, actual demand. Then, I re-run the full cost optimisation using that same tariff to find the new optimal schedule and its cost. Finally, I compile the results into a summary table. This table clearly demonstrates how the percentage of potential cost savings changes as the financial incentive to shift energy usage to cheaper periods becomes stronger. This analysis is vital for understanding the effectiveness of the optimisation strategy under different market conditions.

In [140…

```python
import pandas as pd
import numpy as np
import xgboost as xgb
import pulp
import os

# Setup parameters
# This code assumes 'X_test' and 'y_test' DataFrames are available in memory,
# along with a trained 90th percentile model.

# --- Define the Three TOU Price Scenarios ---
TOU_BASELINE = { # Your original prices
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.40, 17: 0.40, 18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
TOU_LOW_DIFFERENTIAL = { # Peak is cheaper, off-peak is more expensive (less inc
    0: 0.15, 1: 0.15, 2: 0.15, 3: 0.15, 4: 0.15, 5: 0.15, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.35, 17: 0.35, 18: 0.35, 19: 0.35, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
```

```
TOU_HIGH_DIFFERENTIAL = { # Peak is more expensive, off-peak is cheaper (more in
    0: 0.10, 1: 0.10, 2: 0.10, 3: 0.10, 4: 0.10, 5: 0.10, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.50, 17: 0.50, 18: 0.50, 19: 0.50, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
DEMAND_CHARGE_POUNDS_PER_KW = 12.00
SAMPLE_DAY = '2019-07-16'
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"

# Analysis Functions

def generate_forecasts(data_source, target_date_str, quantiles_needed=[0.5, 0.9]
    """Generates forecasts for a list of specified quantiles."""
    forecasts = {}
    target_date = pd.to_datetime(target_date_str)
    day_mask = (data_source.index.date == target_date.date())
    X_day = data_source[day_mask]
    for q in quantiles_needed:
        model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(q*10
        model = xgb.XGBRegressor()
        model.load_model(model_path)
        forecasted_demand = model.predict(X_day)
        forecasts[f'q_{q}'] = np.maximum(forecasted_demand, 0)
    return pd.DataFrame(forecasts, index=X_day.index)

def run_optimization_with_tou(forecast_df, tou_prices_dict):
    """Runs optimization using a specific TOU price dictionary."""
    median = forecast_df['q_0.5'].values
    upper_bounds = forecast_df['q_0.9'].values
    total_energy = median.sum()
    num_intervals = len(median)

    prob = pulp.LpProblem("Optimization_TOU", pulp.LpMinimize)
    power = pulp.LpVariable.dicts("P", range(num_intervals), lowBound=0)
    peak = pulp.LpVariable("Peak", lowBound=0)

    tou_prices_list = [tou_prices_dict[i] for i in range(num_intervals)]
    energy_cost = pulp.lpSum(power[i] * tou_prices_list[i] for i in range(num_in
    demand_cost = peak * DEMAND_CHARGE_POUNDS_PER_KW
    prob += energy_cost + demand_cost

    prob += pulp.lpSum(power) >= total_energy
    for i in range(num_intervals):
        prob += power[i] <= upper_bounds[i]
        prob += peak >= power[i]

    prob.solve(pulp.PULP_CBC_CMD(msg=False))
    return [power[i].value() for i in range(num_intervals)] if pulp.LpStatus[pro

def calculate_total_cost(schedule, tou_prices_dict, demand_charge_price):
    """Calculates the total daily cost of a given schedule and tariff."""
    schedule_np = np.array(schedule)
    energy_cost = sum(p * tou_prices_dict[h] for h, p in enumerate(schedule_np))
    peak_demand = np.max(schedule_np) if schedule_np.size > 0 else 0
    demand_cost = peak_demand * demand_charge_price
    return energy_cost + demand_cost

# Main script execution
```

```
if __name__ == "__main__":
    try:
        tou_scenarios = {
            "Low Differential": TOU_LOW_DIFFERENTIAL,
            "Baseline Differential": TOU_BASELINE,
            "High Differential": TOU_HIGH_DIFFERENTIAL
        }
        results_list = []

        # Prepare data ONCE
        actual_demand_day = y_test[y_test.index.date == pd.to_datetime(SAMPLE_DA
        forecast_df = generate_forecasts(X_test, SAMPLE_DAY)

        print("\n--- Running Sensitivity Analysis for TOU Price Differential ---

        for name, tou_prices in tou_scenarios.items():
            print(f"Testing scenario: {name}...")

            # 1. Recalculate the baseline cost for THIS specific tariff
            uncontrolled_cost = calculate_total_cost(actual_demand_day.values, t

            # 2. Run the optimization with THIS specific tariff
            optimised_schedule = run_optimization_with_tou(forecast_df, tou_pric
            optimised_cost = calculate_total_cost(optimised_schedule, tou_prices

            # 3. Calculate the savings against the correct baseline
            savings_pct = ((uncontrolled_cost - optimised_cost) / uncontrolled_c

            results_list.append({
                'TOU Profile': name,
                'Total Optimised Cost (£)': f"£{optimised_cost:,.2f}",
                'Total Savings (%)': f"{savings_pct:.1f}%"
            })

        # --- Display the final results table ---
        sensitivity_df = pd.DataFrame(results_list)
        print("\n" + "="*60)
        print(" TOU PRICE DIFFERENTIAL SENSITIVITY RESULTS")
        print("="*60)
        print(sensitivity_df.to_string(index=False))
        print("="*60)

    except Exception as e:
        print(f"\n SCRIPT FAILED: {e}")
```

```
--- Running Sensitivity Analysis for TOU Price Differential ---
Testing scenario: Low Differential...
Testing scenario: Baseline Differential...
Testing scenario: High Differential...

============================================================
 TOU PRICE DIFFERENTIAL SENSITIVITY RESULTS
============================================================
          TOU Profile Total Optimised Cost (£) Total Savings (%)
     Low Differential                 £741.64             60.1%
Baseline Differential                 £749.71             59.9%
    High Differential                 £767.55             59.4%
============================================================
```

Sensitivity Analysis of Rule-Based Charging

To provide a fair comparison with my forecast-informed approach, I am now performing a sensitivity analysis on a simpler, rule-based charging strategy.

First, I define the logic for this strategy: for any given time-of-use (TOU) tariff, the simulation identifies the hours with the absolute lowest electricity price and then shifts the entire day's energy demand to be spread evenly across only those cheapest hours.

In the main part of the script, I loop through the same three TOU pricing scenarios as before (low, baseline, and high differential). For each scenario, I calculate the baseline cost of uncontrolled charging and the cost of the rule-based schedule. Finally, I compile the results into a summary table. This analysis clearly demonstrates how the effectiveness of a simple, price-chasing strategy is directly influenced by the financial incentive offered by the tariff structure, providing a crucial point of comparison for the more advanced, forecast-informed optimisation.

In [141…

```
import pandas as pd
import numpy as np
import pulp

# Setup & Parameters
# This code assumes 'y_test' DataFrame is available in your notebook.

TOU_BASELINE = {
    0: 0.12, 1: 0.12, 2: 0.12, 3: 0.12, 4: 0.12, 5: 0.12, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.40, 17: 0.40, 18: 0.40, 19: 0.40, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
TOU_LOW_DIFFERENTIAL = {
    0: 0.15, 1: 0.15, 2: 0.15, 3: 0.15, 4: 0.15, 5: 0.15, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.35, 17: 0.35, 18: 0.35, 19: 0.35, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
TOU_HIGH_DIFFERENTIAL = {
    0: 0.10, 1: 0.10, 2: 0.10, 3: 0.10, 4: 0.10, 5: 0.10, 6: 0.22, 7: 0.22,
    8: 0.22, 9: 0.22, 10: 0.22, 11: 0.22, 12: 0.22, 13: 0.22, 14: 0.22,
    15: 0.22, 16: 0.50, 17: 0.50, 18: 0.50, 19: 0.50, 20: 0.22, 21: 0.22,
    22: 0.22, 23: 0.22
}
DEMAND_CHARGE_POUNDS_PER_KW = 12.00
SAMPLE_DAY = '2019-07-16'

# Analysis function

def simulate_rule_based_charging(total_energy_needed, tou_prices_dict):
    """
    Simulates charging by spreading demand evenly across the cheapest hours.
    """
    num_intervals = 24

    # Find the lowest price in the tariff
```

```
    min_price = min(tou_prices_dict.values())

    # Identify all hours that have this lowest price
    cheapest_hours = [hour for hour, price in tou_prices_dict.items() if price =

    # Spread the load evenly across these cheapest hours
    power_per_hour = total_energy_needed / len(cheapest_hours) if cheapest_hours

    schedule = [0] * num_intervals
    for hour in cheapest_hours:
        schedule[hour] = power_per_hour

    return schedule

def calculate_total_cost(schedule, tou_prices_dict, demand_charge_price):
    """Calculates the total daily cost of a given schedule and tariff."""
    schedule_np = np.array(schedule)
    energy_cost = sum(p * tou_prices_dict[h] for h, p in enumerate(schedule_np))
    peak_demand = np.max(schedule_np) if schedule_np.size > 0 else 0
    demand_cost = peak_demand * demand_charge_price
    return energy_cost + demand_cost

# main execution

if __name__ == "__main__":
    try:
        tou_scenarios = {
            "Low Differential": TOU_LOW_DIFFERENTIAL,
            "Baseline Differential": TOU_BASELINE,
            "High Differential": TOU_HIGH_DIFFERENTIAL
        }
        results_list = []

        # Prepare data ONCE
        actual_demand_day = y_test[y_test.index.date == pd.to_datetime(SAMPLE_DA
        total_energy_needed = actual_demand_day.sum()

        print("\n--- Running Sensitivity Analysis for Rule-Based Charging ---")

        for name, tou_prices in tou_scenarios.items():
            print(f"Testing scenario: {name}...")

            # 1. Recalculate the baseline cost for THIS specific tariff
            uncontrolled_cost = calculate_total_cost(actual_demand_day.values, t

            # 2. Run the RULE-BASED simulation with THIS specific tariff
            rule_based_schedule = simulate_rule_based_charging(total_energy_need
            rule_based_cost = calculate_total_cost(rule_based_schedule, tou_pric

            # 3. Calculate the savings against the correct baseline
            savings_pct = ((uncontrolled_cost - rule_based_cost) / uncontrolled_

            results_list.append({
                'TOU Profile': name,
                'Total Rule-Based Cost (£)': f"£{rule_based_cost:,.2f}",
                'Total Savings (%)': f"{savings_pct:.1f}%"
            })

        # --- Display the final results table ---
        sensitivity_df = pd.DataFrame(results_list)
```

```
        print("\n" + "="*60)
        print(" RULE-BASED CHARGING SENSITIVITY RESULTS")
        print("="*60)
        print(sensitivity_df.to_string(index=False))
        print("="*60)

    except Exception as e:
        print(f"\n SCRIPT FAILED: {e}")
```

```
--- Running Sensitivity Analysis for Rule-Based Charging ---
Testing scenario: Low Differential...
Testing scenario: Baseline Differential...
Testing scenario: High Differential...

============================================================
 RULE-BASED CHARGING SENSITIVITY RESULTS
============================================================
          TOU Profile Total Rule-Based Cost (£) Total Savings (%)
     Low Differential                 £1,922.51             -3.5%
Baseline Differential                 £1,895.69             -1.4%
    High Differential                 £1,877.81              0.7%
============================================================
```

Evaluating Forecast Calibration

To ensure my uncertainty predictions are reliable, I am now formally evaluating the calibration of my probabilistic forecasts. This process checks whether the predicted uncertainty range accurately reflects the true uncertainty in the real-world data.

First, I define several functions to calculate key performance metrics for probabilistic forecasts. This includes the pinball loss, which is a specialised error metric for quantile regression, and the empirical coverage, which measures how often the actual energy demand falls within my 10th-90th percentile prediction interval. I also create a function to generate a reliability curve, which provides a clear visual assessment of how well-calibrated the forecasts are.

In the main execution block, I first load my trained quantile models and generate forecasts for the entire test set. I then use my helper functions to calculate and print the final calibration metrics. The goal is for the empirical coverage to be as close as possible to the nominal coverage of 80%, which would indicate that the model's representation of uncertainty is highly reliable. Finally, I generate the reliability curve plot for a clear visual confirmation.

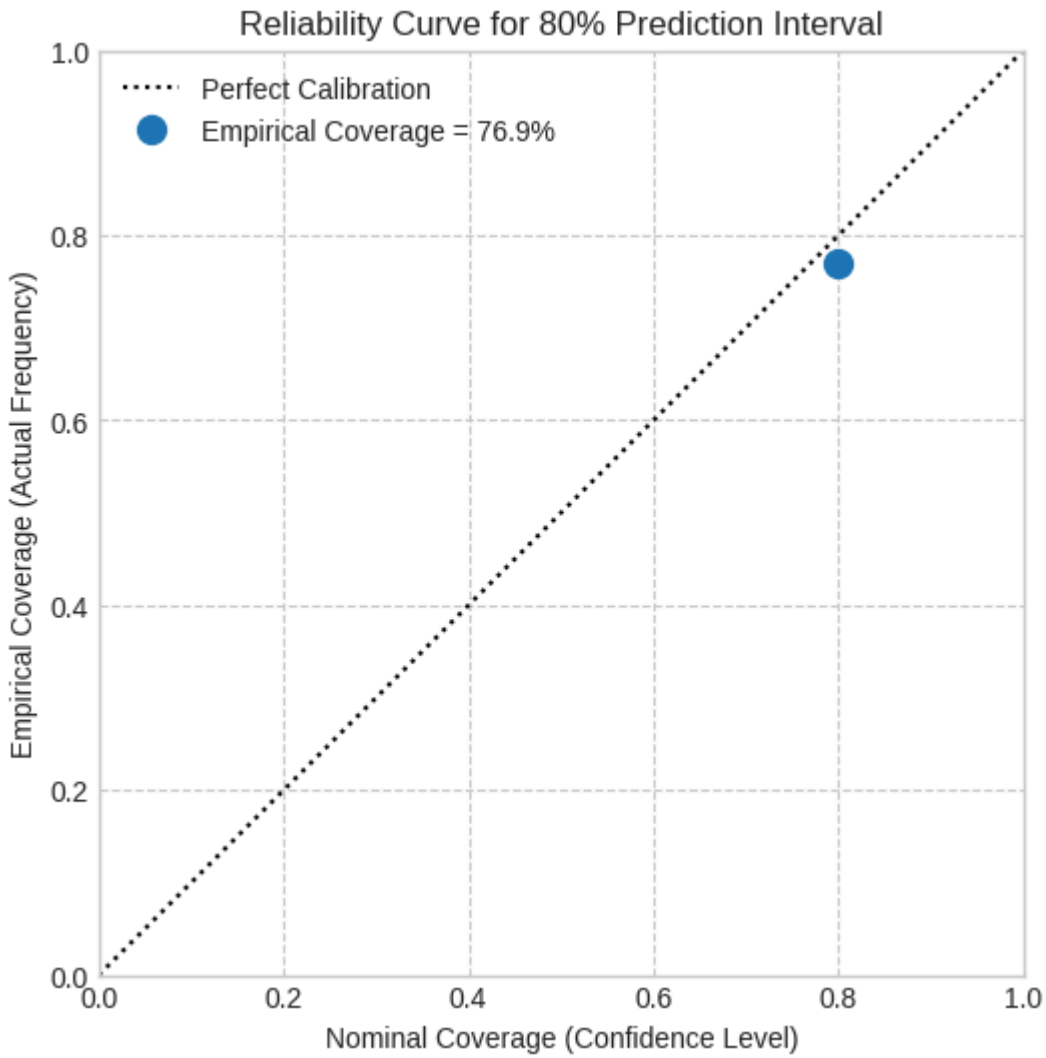
In [144…

```
import pandas as pd
import numpy as np
import xgboost as xgb
import os
import matplotlib.pyplot as plt

# Setup parameters
# This assumes X_test and y_test are available in memory.
MODELS_DIR = "/content/keras_tuner_dir/ev_demand_forecasting"
QUANTILES = [0.10, 0.50, 0.90]

# ============================================
```

```
# 2. CREATE THE FORECASTS DATAFRAME
# ==========================================
# This section was missing. It loads your pre-trained models to create the forec
print("--- Loading pre-trained quantile models and generating forecasts ---")
forecast_data = {}
for q in QUANTILES:
    model_path = os.path.join(MODELS_DIR, f'ev_demand_xgb_quantile_{int(q*100)}.
    if not os.path.exists(model_path):
        raise FileNotFoundError(f"Model file not found: {model_path}")

    model = xgb.XGBRegressor()
    model.load_model(model_path)
    forecast_data[f'q_{q}'] = model.predict(X_test)

test_forecasts = pd.DataFrame(forecast_data, index=y_test.index)
print(" Forecasts DataFrame created successfully.")

# Caliberation metric functions

def calculate_pinball_loss(y_true, y_pred, quantile):
    """Calculates the pinball loss for a single quantile."""
    error = y_true - y_pred
    return np.mean(np.maximum(quantile * error, (quantile - 1) * error))

def compute_calibration_stats(y_true, forecasts_df):
    """Computes coverage and average pinball loss for 10-90 quantiles."""
    actuals = y_true.values
    q10 = forecasts_df['q_0.1'].values
    q50 = forecasts_df['q_0.5'].values
    q90 = forecasts_df['q_0.9'].values

    is_covered = (actuals >= q10) & (actuals <= q90)
    coverage = np.mean(is_covered) * 100

    loss_q10 = calculate_pinball_loss(actuals, q10, 0.10)
    loss_q50 = calculate_pinball_loss(actuals, q50, 0.50)
    loss_q90 = calculate_pinball_loss(actuals, q90, 0.90)
    avg_pinball_loss = np.mean([loss_q10, loss_q50, loss_q90])

    return {'coverage_80_percent': coverage, 'avg_pinball_loss': avg_pinball_los

def plot_reliability_curve(y_true, forecasts_df):
    """Plots a reliability curve for the 10-90 prediction interval."""
    actuals = y_true.values
    q10 = forecasts_df['q_0.1'].values
    q90 = forecasts_df['q_0.9'].values

    coverage = np.mean((actuals >= q10) & (actuals <= q90))

    plt.figure(figsize=(6, 6))
    plt.plot([0, 1], [0, 1], 'k:', label='Perfect Calibration')
    plt.plot([0.8], [coverage], 'o', markersize=10, label=f'Empirical Coverage =

    plt.title('Reliability Curve for 80% Prediction Interval')
    plt.xlabel('Nominal Coverage (Confidence Level)')
    plt.ylabel('Empirical Coverage (Actual Frequency)')
    plt.xlim(0, 1)
    plt.ylim(0, 1)
    plt.grid(True, linestyle='--')
    plt.legend()
```

```
    plt.gca().set_aspect('equal', adjustable='box')
    plt.savefig("reliability_curve.png", dpi=300)
    plt.show()

# main script execution

if __name__ == "__main__":
    # Calculate stats
    calibration_metrics = compute_calibration_stats(y_test, test_forecasts)

    print("\n" + "="*50)
    print(" PROBABILISTIC FORECAST CALIBRATION RESULTS")
    print("="*50)
    print(f"Target Coverage (10-90 Interval): 80.0%")
    print(f"Empirical Coverage: {calibration_metrics['coverage_80_percent']:.1f}
    print(f"Average Pinball Loss (q10, q50, q90): {calibration_metrics['avg_pinb
    print("="*50)

    # Generate plot
    plot_reliability_curve(y_test, test_forecasts)
```

```
--- Loading pre-trained quantile models and generating forecasts ---
 Forecasts DataFrame created successfully.

==================================================
 PROBABILISTIC FORECAST CALIBRATION RESULTS
==================================================
Target Coverage (10-90 Interval): 80.0%
Empirical Coverage: 76.9%
Average Pinball Loss (q10, q50, q90): 5.1624
==================================================
```



Analysing the Socio-Technical Dimensions

This final script is dedicated to creating the key visualisations for the socio-technical analysis in my dissertation. It moves beyond the purely technical aspects of forecasting and optimisation to explore the human and societal factors that shape the charging network. The script is organised into three main analyses:

First, I define a new data cleaning function to standardise the names of the charging locations. This is a crucial preparatory step to ensure that my analysis of geographic usage is accurate.

The script then generates three specific plots:

- Geographic Equity: This function aggregates the charging data by location to produce a bar chart showing the total energy delivered at each site. This allows me to analyse the distribution of charging infrastructure usage, providing evidence for discussions about the equity of access and the balance between high-capacity urban hubs and smaller, more remote spokes.

- Cost vs. Convenience: This function plots the un-optimised, convenience-driven charging schedule directly against the cost-optimised schedule. By overlaying the time-of-use electricity prices, this visualisation provides a clear illustration of the central behavioural-economic trade-off, showing how the optimisation algorithm must shift demand away from peak times to achieve savings.
- Social Drivers: This final function compares the average hourly demand profiles for weekdays versus weekends. This plot highlights how predictable social routines, such as the typical work week, are a primary driver of the charging patterns that the forecasting model must learn.

In [145…

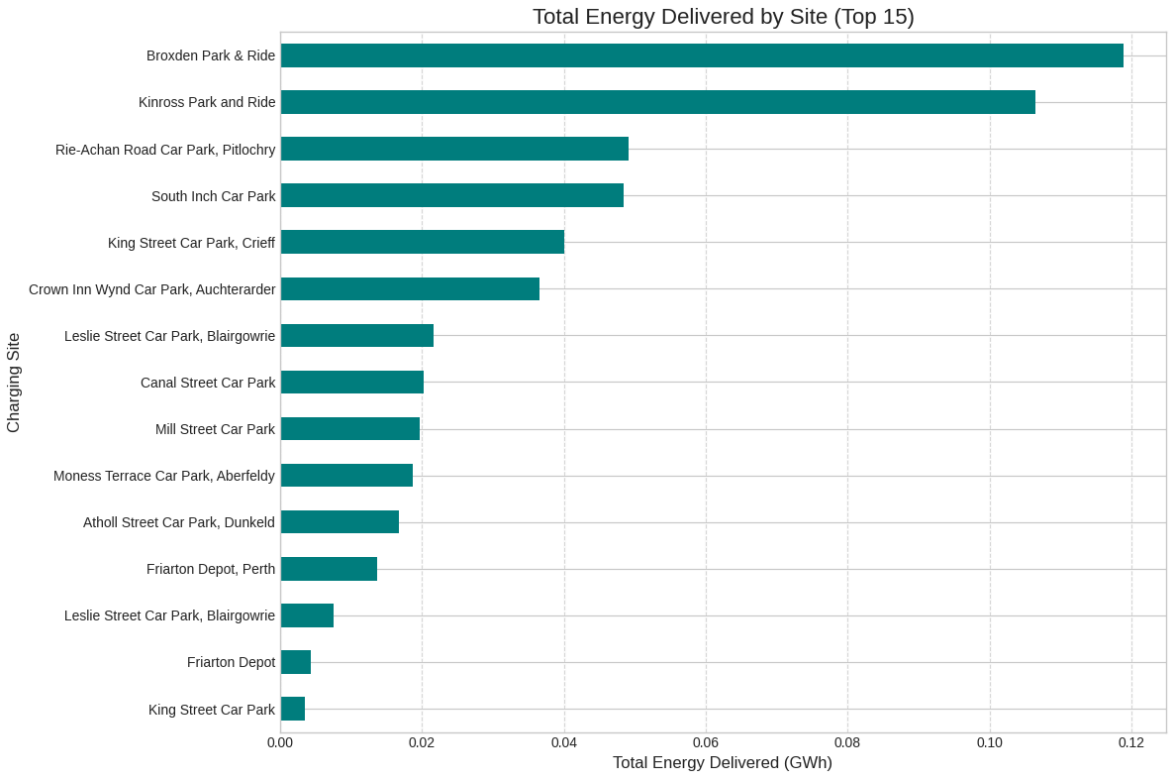
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Setup

# Define the directory where all plots will be saved
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)
print(f"All socio-technical visualizations will be saved to the '{SAVE_DIR}' dir

# --- Prerequisites ---
# This script assumes the following variables are available in your notebook's m
# - df: The original, cleaned DataFrame with individual charging session data.
# - df_hourly: The hourly aggregated DataFrame.
# - weekly_schedules: The dictionary containing the three strategy schedules for
# - TOU_PRICES_POUNDS_DICT: The dictionary of hourly prices.

# Data cleaning function

def clean_site_names(df_sessions):
    """
    Standardizes charging site names to merge duplicates.
    e.g., 'Broxden Park & Ride, Perth' becomes 'Broxden Park & Ride'.
    """
    print("\n--- Cleaning and standardizing site names for consistent analysis -

    # Define the mapping from old, inconsistent names to a standard name
    name_mapping = {
        'Broxden Park & Ride, Perth': 'Broxden Park & Ride',
        'Kinross Park and Ride, Kinross': 'Kinross Park and Ride',
        'South Inch Car Park, Perth': 'South Inch Car Park',
        'Canal Street Car Park 3rd floor, Perth': 'Canal Street Car Park',
        'Mill Street, Perth': 'Mill Street Car Park' # Standardizing based on co
    }

    # Apply the mapping to the 'Site' column
    df_sessions['Site'] = df_sessions['Site'].replace(name_mapping)
    print("✅ Site names cleaned successfully.")
    return df_sessions


# Plotting functions
```

```
def plot_geographic_equity(df_sessions, save_dir):
    """
    Analyzes and plots charging patterns by location to assess equity of access.
    """
    print("\n--- Generating Plot 1: Geographic Equity Analysis ---")
    try:
        # Use the standardized column name 'CP_ID'
        if 'CP_ID' in df_sessions.columns:
            cpid_col = 'CP_ID'
        else:
            raise KeyError("Could not find a charge point ID column ('CP_ID').")

        site_usage = df_sessions.groupby('Site').agg(
            TotalEnergy_GWh=('Total_kWh', lambda x: x.sum() / 1000000),
            AverageDuration_Hrs=('ChargingDurationHrs', 'mean'),
            NumberOfSessions=(cpid_col, 'count')
        ).sort_values(by='TotalEnergy_GWh', ascending=False)

        # Plot 1: Total Energy Delivered per Site
        plt.style.use('seaborn-v0_8-whitegrid')
        fig, ax = plt.subplots(figsize=(12, 8))

        top_n = 15
        site_usage.head(top_n).sort_values(by='TotalEnergy_GWh', ascending=True)
            kind='barh', ax=ax, color='teal'
        )

        ax.set_title(f'Total Energy Delivered by Site (Top {top_n})', fontsize=1
        ax.set_xlabel('Total Energy Delivered (GWh)', fontsize=12)
        ax.set_ylabel('Charging Site', fontsize=12)
        ax.grid(axis='x', linestyle='--', alpha=0.7)

        plt.tight_layout()
        save_path = os.path.join(save_dir, "socio_geographic_equity.png")
        plt.savefig(save_path)
        plt.show()
        plt.close()
        print(f"Saved: socio_geographic_equity.png")

    except Exception as e:
        print(f"Could not generate Geographic Equity plot. Error: {e}")


def plot_cost_vs_convenience(schedules, tou_prices, save_dir):
    """
    Visualizes the trade-off between the cost-optimized schedule and user conven
    """
    print("\n--- Generating Plot 2: Behavioral Economics (Cost vs. Convenience)
    try:
        uncontrolled = schedules['Uncontrolled'][:24]
        forecast_informed = schedules['Forecast-Informed'][:24]
        tou_prices_list = [tou_prices[h] for h in range(24)]

        plt.style.use('seaborn-v0_8-whitegrid')
        fig, ax1 = plt.subplots(figsize=(14, 7))

        hours = np.arange(24)

        ax1.plot(hours, uncontrolled, 'o--', color='black', label='Uncontrolled
```

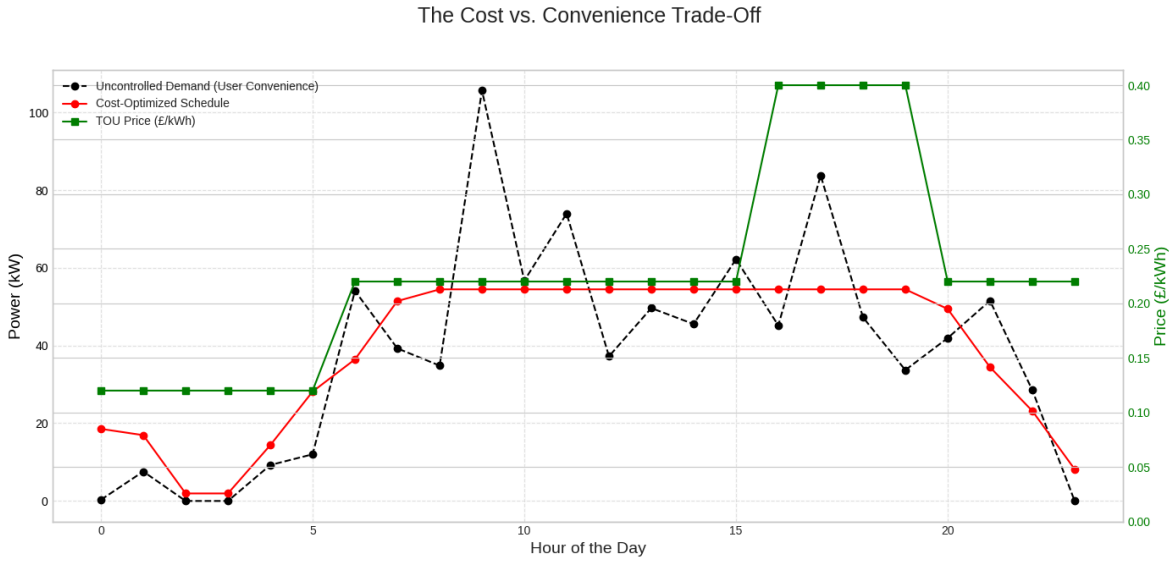
```
        ax1.plot(hours, forecast_informed, 'o-', color='red', label='Cost-Optimi
        ax1.set_xlabel('Hour of the Day', fontsize=14)
        ax1.set_ylabel('Power (kW)', fontsize=14, color='black')
        ax1.tick_params(axis='y', labelcolor='black')
        ax1.grid(True, which='both', linestyle='--', alpha=0.5)

        ax2 = ax1.twinx()
        ax2.plot(hours, tou_prices_list, 's-', color='green', label='TOU Price (
        ax2.set_ylabel('Price (£/kWh)', fontsize=14, color='green')
        ax2.tick_params(axis='y', labelcolor='green')
        ax2.set_ylim(bottom=0)

        fig.suptitle('The Cost vs. Convenience Trade-Off', fontsize=18)
        lines, labels = ax1.get_legend_handles_labels()
        lines2, labels2 = ax2.get_legend_handles_labels()
        ax2.legend(lines + lines2, labels + labels2, loc='upper left')

        plt.tight_layout(rect=[0, 0.03, 1, 0.95])
        save_path = os.path.join(save_dir, "socio_cost_vs_convenience.png")
        plt.savefig(save_path)
        plt.show()
        plt.close()
        print(f"Saved: socio_cost_vs_convenience.png")

    except Exception as e:
        print(f"Could not generate Cost vs. Convenience plot. Error: {e}")


def plot_social_drivers(df_hourly, save_dir):
    """
    Plots and compares the average demand profiles for weekdays vs. weekends.
    """
    print("\n--- Generating Plot 3: Social Drivers (Weekday vs. Weekend) ---")
    try:
        target_col = 'Total_kwh'
        df_hourly['DayType'] = df_hourly['WorkingStatus'].apply(lambda x: 'Weekd
        demand_profiles = df_hourly.groupby(['DayType', 'ConnectionHour'])[targe

        plt.style.use('seaborn-v0_8-whitegrid')
        plt.figure(figsize=(12, 6))

        plt.plot(demand_profiles.index, demand_profiles['Weekday'], 'o-', label=
        plt.plot(demand_profiles.index, demand_profiles['Weekend'], 's-', label=

        plt.title('Charging Demand Driven by Social Routines', fontsize=16)
        plt.xlabel('Hour of the Day', fontsize=12)
        plt.ylabel('Average Hourly Demand (kWh)', fontsize=12)
        plt.xticks(np.arange(0, 24, 2))
        plt.legend(fontsize='large')
        plt.grid(True, which='both', linestyle='--', alpha=0.7)

        plt.tight_layout()
        save_path = os.path.join(save_dir, "socio_weekday_vs_weekend.png")
        plt.savefig(save_path)
        plt.show()
        plt.close()
        print(f"Saved: socio_weekday_vs_weekend.png")

    except Exception as e:
        print(f"Could not generate Social Drivers plot. Error: {e}")
```

```
# Main execution

if __name__ == "__main__":
    print("\nStarting socio-technical visualization export process...")

    required_vars = ['df', 'df_hourly', 'weekly_schedules', 'TOU_PRICES_POUNDS_D
    missing_vars = [var for var in required_vars if var not in locals() and var

    if not missing_vars:
        # --- NEW STEP: Clean the site names in the 'df' DataFrame first ---
        df_cleaned = clean_site_names(df.copy()) # Use a copy to avoid modifying

        # Run all plotting functions with the cleaned data
        plot_geographic_equity(df_cleaned, SAVE_DIR)
        plot_cost_vs_convenience(weekly_schedules, TOU_PRICES_POUNDS_DICT, SAVE_
        plot_social_drivers(df_hourly, SAVE_DIR)
        print("\n Socio-technical visualization export process complete!")
    else:
        print(f"\n Could not run script. The following variables are missing: {'
        print("Please ensure you have run all previous notebook cells to generat
```

```
All socio-technical visualizations will be saved to the 'project_visualizations'
directory.

Starting socio-technical visualization export process...

--- Cleaning and standardizing site names for consistent analysis ---
✅ Site names cleaned successfully.

--- Generating Plot 1: Geographic Equity Analysis ---
```




```
Saved: socio_geographic_equity.png

--- Generating Plot 2: Behavioral Economics (Cost vs. Convenience) ---
```




```
Saved: socio_cost_vs_convenience.png

--- Generating Plot 3: Social Drivers (Weekday vs. Weekend) ---
Could not generate Social Drivers plot. Error: 'Column not found: Total_kwh'

 Socio-technical visualization export process complete!
```

In [147…

```
!pip install contextily pyproj adjustText
```

```
Collecting contextily
  Downloading contextily-1.6.2-py3-none-any.whl.metadata (2.9 kB)
Collecting pyproj
  Downloading pyproj-3.7.2-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (31 kB)
Collecting adjustText
  Downloading adjustText-1.3.0-py3-none-any.whl.metadata (3.1 kB)
Collecting geopy (from contextily)
  Downloading geopy-2.4.1-py3-none-any.whl.metadata (6.8 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packa
ges (from contextily) (3.10.0)
Collecting mercantile (from contextily)
  Downloading mercantile-1.2.1-py3-none-any.whl.metadata (4.8 kB)
Requirement already satisfied: pillow in /usr/local/lib/python3.11/dist-packages
(from contextily) (11.3.0)
Collecting rasterio (from contextily)
  Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_
64.whl.metadata (9.1 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-package
s (from contextily) (2.32.3)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages
(from contextily) (1.5.1)
Collecting xyzservices (from contextily)
  Downloading xyzservices-2025.4.0-py3-none-any.whl.metadata (4.3 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages
(from pyproj) (2025.8.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from adjustText) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages
(from adjustText) (1.16.1)
Collecting geographiclib<3,>=1.52 (from geopy->contextily)
  Downloading geographiclib-2.0-py3-none-any.whl.metadata (1.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib->contextily) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pac
kages (from matplotlib->contextily) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib->contextily) (4.59.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dis
t-packages (from matplotlib->contextily) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-
packages (from matplotlib->contextily) (25.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist
-packages (from matplotlib->contextily) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/
dist-packages (from matplotlib->contextily) (2.9.0.post0)
Requirement already satisfied: click>=3.0 in /usr/local/lib/python3.11/dist-packa
ges (from mercantile->contextily) (8.2.1)
Collecting affine (from rasterio->contextily)
  Downloading affine-2.4.0-py3-none-any.whl.metadata (4.0 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.11/dist-packages
(from rasterio->contextily) (25.3.0)
Collecting cligj>=0.5 (from rasterio->contextily)
  Downloading cligj-0.7.2-py3-none-any.whl.metadata (5.0 kB)
Collecting click-plugins (from rasterio->contextily)
  Downloading click_plugins-1.1.1.2-py2.py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python
3.11/dist-packages (from requests->contextily) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-pac
kages (from requests->contextily) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/di
```

```
st-packages (from requests->contextily) (2.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil>=2.7->matplotlib->contextily) (1.17.0)
Downloading contextily-1.6.2-py3-none-any.whl (17 kB)
Downloading pyproj-3.7.2-cp311-cp311-manylinux_2_28_x86_64.whl (9.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 84.0 MB/s eta 0:00:00
Downloading adjustText-1.3.0-py3-none-any.whl (13 kB)
Downloading geopy-2.4.1-py3-none-any.whl (125 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 125.4/125.4 kB 12.3 MB/s eta 0:00:00
Downloading mercantile-1.2.1-py3-none-any.whl (14 kB)
Downloading rasterio-1.4.3-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (22.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 22.2/22.2 MB 90.6 MB/s eta 0:00:00
Downloading xyzservices-2025.4.0-py3-none-any.whl (90 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 90.4/90.4 kB 7.5 MB/s eta 0:00:00
Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Downloading geographiclib-2.0-py3-none-any.whl (40 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 40.3/40.3 kB 3.2 MB/s eta 0:00:00
Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Downloading click_plugins-1.1.1.2-py2.py3-none-any.whl (11 kB)
Installing collected packages: xyzservices, pyproj, mercantile, geographiclib, cl
igj, click-plugins, affine, rasterio, geopy, contextily, adjustText
Successfully installed adjustText-1.3.0 affine-2.4.0 click-plugins-1.1.1.2 cligj-
0.7.2 contextily-1.6.2 geographiclib-2.0 geopy-2.4.1 mercantile-1.2.1 pyproj-3.7.
2 rasterio-1.4.3 xyzservices-2025.4.0
```

Downloading Geographic Boundary Data for PKC

To create the choropleth map for my geographic analysis, I first need the map data itself. This script automates the process of downloading the necessary geographic boundary file.

I am using the requests library to fetch a GeoJSON file from a stable online source. This file contains the specific shapes of all the administrative wards in Scotland. The script makes a network request to the URL, checks to ensure the download is successful, and then saves the content to a local file named ward_boundaries.geojson. I have also included error handling to manage any potential network problems that might occur during the download. This step is a prerequisite for creating the final map visualisation.

In [148…

```python
import requests
import os

# This is a direct link to a raw GeoJSON file and is much more reliable.
wards_file_url = "https://raw.githubusercontent.com/martinjc/UK-GeoJSON/master/j
# The local file path where the data will be saved
local_file_path = "ward_boundaries.geojson"

print(f"--- Downloading ward boundary data from new, stable URL ---")
print(f"Source: {wards_file_url}")

try:
    # Make the request to download the file with a timeout
    response = requests.get(wards_file_url, timeout=60)
```

```
    # Check if the request was successful (e.g., not a 404 or 500 error)
    response.raise_for_status()

    # Save the downloaded content to the local file
    with open(local_file_path, 'wb') as f:
        f.write(response.content)

    print(f"\n Successfully downloaded and saved data to: {local_file_path}")
    print("You can now proceed to run the 'Choropleth Map' script.")

except requests.exceptions.RequestException as e:
    print(f"\n FAILED to download the file. A network error occurred: {e}")
    print("If this error persists, there may be a broader network issue. Please
```

```
--- Downloading ward boundary data from new, stable URL ---
Source: https://raw.githubusercontent.com/martinjc/UK-GeoJSON/master/json/adminis
trative/sco/lad.json

 Successfully downloaded and saved data to: ward_boundaries.geojson
You can now proceed to run the 'Choropleth Map' script.
```

Generating the Final Geographic Usage Map

This final script is dedicated to creating the choropleth map, which is the key visualisation for the socio-technical analysis of geographic equity. The process is broken down into two main stages: data preparation and map generation.

First, in the data preparation stage, I create a function that takes my raw session data and merges it with an official list of charger locations. This is a critical data wrangling step that involves several transformations:

- I load the official location data, which contains the coordinates for each charging point.
- I perform an essential coordinate transformation, converting the locations from the British National Grid system to the standard WGS84 format used by most mapping software.
- I then carry out a definitive manual mapping to standardise the site names, as they are often inconsistent between the session data and the official location list. This ensures that the usage data is correctly joined to its geographic coordinates.
- Finally, I merge the two datasets to create a single, unified GeoDataFrame that contains both the usage statistics and the precise location for each charging site.

In the second stage, a dedicated map generation function takes this prepared data and creates the final visualisation.

- It begins by loading the previously downloaded GeoJSON file to create a base map showing the administrative boundary of Perth and Kinross.
- The charger locations are then plotted on top of this map as a scatter plot. I use visual encoding to convey two types of information simultaneously: the size of each

point represents the total number of charging sessions, while the colour represents the total energy delivered.

- To ensure the map is readable, I add a numbered label to each point and create a custom legend box directly on the map. This legend clearly lists the name of each site corresponding to its number, which is a much more effective solution than a standard legend for a map with this many data points.

- The final output is a high-resolution, data-rich map that provides a clear and immediate understanding of the geographic distribution and usage patterns of the entire charging network, saved as a PNG file for inclusion in my dissertation.

In [152…

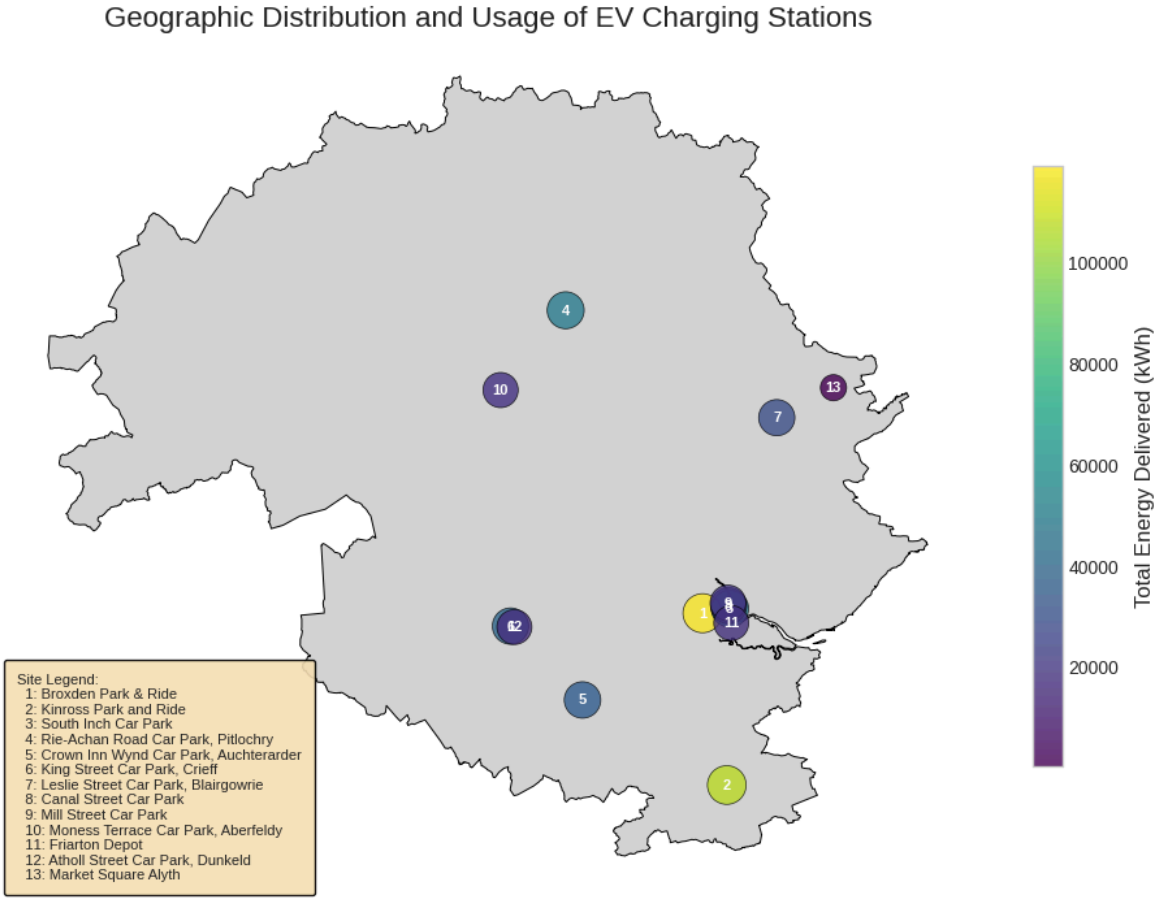
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import geopandas as gpd
from pyproj import Transformer
from adjustText import adjust_text
from matplotlib.offsetbox import AnchoredText

# setup

# Define the directory where the map will be saved
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)
print(f"The final map will be saved to the '{SAVE_DIR}' directory.")

# --- Prerequisites ---
# This script assumes the 'df' DataFrame is available in your notebook's memory.

# Data preparation and merging

def prepare_map_data(df_sessions, locations_csv_path):
    """
    Cleans, merges, and transforms coordinate data for mapping.
    """
    print("\n--- Preparing data for mapping ---")
    try:
        # Load the official PKC charger locations
        df_locations = pd.read_csv(locations_csv_path)

        # --- Rename the site and coordinate columns to a standard format ---
        df_locations.rename(columns={'CP_Name': 'Site', 'x': 'longitude', 'y': '

        # --- Force data types and handle missing values ---
        df_locations['Site'] = df_locations['Site'].astype(str).str.strip()
        df_sessions['Site'] = df_sessions['Site'].astype(str).str.strip()
        df_locations['latitude'] = pd.to_numeric(df_locations['latitude'], error
        df_locations['longitude'] = pd.to_numeric(df_locations['longitude'], err
        df_locations.dropna(subset=['latitude', 'longitude'], inplace=True)

        # --- Coordinate Transformation (British National Grid to WGS84) ---
        transformer = Transformer.from_crs("EPSG:27700", "EPSG:4326", always_xy=
        df_locations['lon_wgs84'], df_locations['lat_wgs84'] = transformer.trans
            df_locations['longitude'].values,
            df_locations['latitude'].values
```

```
        )
        print(f" Loaded and processed {len(df_locations)} charger locations.")

        # --- Definitive Manual Mapping ---
        location_name_map = {
            'ACCESS ROAD INTO PARK AND RIDE BROXDEN AVENUE': 'Broxden Park & Rid
            'CAR PARK OFF JUNCTION ROAD PARK AND RIDE': 'Kinross Park and Ride',
            'CAR PARK OFF SHORE ROAD AT SOUTH INCH': 'South Inch Car Park',
            'CANAL STREETMULTY STOREY CAR PARK': 'Canal Street Car Park',
            'CAR PARK MULTI STOREY OFF CANAL STREET': 'Canal Street Car Park',
            'CAR PARK OFF KING STREET TOWARDS GALVELMORE STREET': 'King Street C
            'CAR PARK OFF HIGH STREET': 'Crown Inn Wynd Car Park, Auchterarder',
            'CAR PARK OFF CROFT LANE AT LESLIE STREET': 'Leslie Street Car Park,
            'CAR PARK OFF RIE-ACHAN ROAD': 'Rie-Achan Road Car Park, Pitlochry',
            'CAR PARK OFF MILL STREET EAST': 'Mill Street Car Park',
            'CAR PARK OFF MONESS TERRACE': 'Moness Terrace Car Park, Aberfeldy',
            'FRIARTON WASTE TRANSFER STATION': 'Friarton Depot',
            'CAR PARK IN MARKET SQUARE': 'Market Square Alyth',
            'LEADENFLOWER CAR PARK': 'Atholl Street Car Park, Dunkeld'
        }
        df_locations['Site'] = df_locations['Site'].replace(location_name_map)

        session_name_map = {
            'Broxden Park & Ride, Perth': 'Broxden Park & Ride',
            'Kinross Park and Ride, Kinross': 'Kinross Park and Ride',
            'South Inch Car Park, Perth': 'South Inch Car Park',
            'Canal Street Car Park 3rd floor, Perth': 'Canal Street Car Park',
            'Canal Street Car Park 3rd floor': 'Canal Street Car Park',
            'Mill Street, Perth': 'Mill Street Car Park',
            'Friarton Depot, Perth': 'Friarton Depot'
        }
        df_sessions['Site'] = df_sessions['Site'].replace(session_name_map)

        # Calculate usage statistics per site
        site_usage = df_sessions.groupby('Site').agg(
            TotalEnergy_kWh=('Total_kWh', 'sum'),
            NumberOfSessions=('CP_ID', 'count')
        ).reset_index()

        # Merge the usage data with the location data
        merged_data = pd.merge(site_usage, df_locations[['Site', 'lat_wgs84', 'l
        merged_data.drop_duplicates(subset='Site', inplace=True)

        print(f" Successfully merged usage data with {len(merged_data)} geolocat
        return merged_data

    except Exception as e:
        print(f" Could not prepare map data. Error: {e}")
        return None

# Final Map generation with legends

def create_final_map(map_data, boundary_file_path, save_dir):
    """
    Generates and saves a static map with a real boundary background and a numbe
    """
    print("\n--- Generating final map visualization with numbered legend ---")
    try:
        # --- Load and prepare the boundary data ---
        gdf_boundaries = gpd.read_file(boundary_file_path)
```

```
        council_name_col = 'LAD13NM'
        pkc_boundary = gdf_boundaries[gdf_boundaries[council_name_col] == 'Perth

        # --- Convert charger data to a GeoDataFrame ---
        gdf_chargers = gpd.GeoDataFrame(
            map_data,
            geometry=gpd.points_from_xy(map_data.lon_wgs84, map_data.lat_wgs84),
            crs="EPSG:4326"
        ).sort_values(by='NumberOfSessions', ascending=False).reset_index(drop=T

        # --- Plotting ---
        plt.style.use('seaborn-v0_8-whitegrid')
        fig, ax = plt.subplots(1, 1, figsize=(12, 12))

        # Plot the main PKC boundary
        pkc_boundary.plot(ax=ax, color='lightgray', edgecolor='black', linewidth

        # Overlay all charger locations
        scatter = ax.scatter(
            gdf_chargers.geometry.x, gdf_chargers.geometry.y,
            s=np.log(gdf_chargers['NumberOfSessions'] + 1) * 50,
            c=gdf_chargers['TotalEnergy_kWh'],
            cmap='viridis', alpha=0.8, edgecolors='k', linewidth=0.5, zorder=10
        )
        cbar = plt.colorbar(scatter, ax=ax, shrink=0.5)
        cbar.set_label('Total Energy Delivered (kWh)', fontsize=12)

        # --- Add Numbered Labels to Map and Create Legend Text ---
        legend_texts = ["Site Legend:"]
        texts_to_adjust = []
        for i, (idx, site) in enumerate(gdf_chargers.iterrows()):
            label_num = i + 1
            texts_to_adjust.append(
                ax.text(site.geometry.x, site.geometry.y, str(label_num), color=
                        ha='center', va='center', fontsize=8, weight='bold', zor
            )
            legend_texts.append(f"  {label_num}: {site['Site']}")

        # Add the legend box to the main plot
        legend_box = AnchoredText("\n".join(legend_texts), loc='lower left', fra
                                  prop=dict(size=8, ha='left'))
        legend_box.patch.set_boxstyle("round,pad=0.5,rounding_size=0.2")
        legend_box.patch.set_facecolor("wheat")
        legend_box.patch.set_alpha(0.9)
        ax.add_artist(legend_box)

        ax.set_title('Geographic Distribution and Usage of EV Charging Stations'
        ax.set_axis_off()

        # Save the map
        map_path = os.path.join(save_dir, "pkc_final_usage_map_with_legend.png")
        plt.savefig(map_path, dpi=1200, bbox_inches='tight')
        plt.show()
        plt.close()

        print(f"\n Final map with numbered legend saved successfully to: {map_pa

    except Exception as e:
        print(f" Could not create the final map. Error: {e}")
```

```
# Main execution

if __name__ == "__main__":
    locations_file = "/content/drive/MyDrive/CIM/Dissertation/Perth and Kinross
    local_wards_file = "ward_boundaries.geojson"

    if 'df' in locals() or 'df' in globals():
        if os.path.exists(locations_file) and os.path.exists(local_wards_file):
            map_data = prepare_map_data(df, locations_file)
            if map_data is not None and not map_data.empty:
                create_final_map(map_data, local_wards_file, SAVE_DIR)
            elif map_data is not None and map_data.empty:
                print("  Warning: No sites could be matched between usage data a
        else:
            print(f" ERROR: Required file not found. Please ensure both '{locati
    else:
        print(" ERROR: The main 'df' DataFrame is not available in memory. Pleas
```

```
The final map will be saved to the 'project_visualizations' directory.

--- Preparing data for mapping ---
 Loaded and processed 45 charger locations.
 Successfully merged usage data with 13 geolocated sites.

--- Generating final map visualization with numbered legend ---
```




```
 Final map with numbered legend saved successfully to: project_visualizations/pkc
_final_usage_map_with_legend.png
```

Loading the Charger Location Data

To get the geographic coordinates for each charging station, I am now loading an official dataset provided by the council which lists all public charging points. I load this data from its CSV file into a new pandas DataFrame called df_locations. Immediately after loading, I inspect the first ten rows to understand its structure and then print a summary of all the columns and their data types to ensure the data has been imported correctly.

In [153…

```
import pandas as pd

# Define the correct file path to your dataset
locations_file_path = "/content/drive/MyDrive/CIM/Dissertation/Perth and Kinross

# Load the dataset into a pandas DataFrame
print(f"--- Loading data from: {locations_file_path} ---")
df_locations = pd.read_csv(locations_file_path)

# --- Preview the data ---

# Display the first 5 rows to see the structure
print("\n--- First 5 Rows of the Dataset ---")
print(df_locations.head(10))

# Display a summary of all columns and their data types
print("\n--- Dataset Info (Columns and Data Types) ---")
df_locations.info()
```

```
--- Loading data from: /content/drive/MyDrive/CIM/Dissertation/Perth and Kinross
Council, UK (Public Charging)/Electric_Vehicle_Charging_Points_276533630659100621
6.csv ---

--- First 5 Rows of the Dataset ---
   MI_PRINX       Id                                              CP_Name         A
DDRESS CH_SPEED  \
0         1  50995.0                       CAR PARK OFF MONESS TERRACE  MONESS T
ERRACE    RAPID
1         2  50811.0                       CAR PARK OFF MONESS TERRACE  MONESS T
ERRACE     FAST
2         3  51261.0                         CAR PARK IN MARKET SQUARE   MARKET
SQUARE     FAST
3         4  51250.0                          CAR PARK OFF HIGH STREET  CROWN IN
N WYND    RAPID
4         5  50994.0           CAR PARK OFF CROFT LANE AT LESLIE STREET   LESLIE
STREET    RAPID
5         6  50993.0  CAR PARK OFF KING STREET TOWARDS GALVELMORE ST...     KING
STREET    RAPID
6         7  50813.0  CAR PARK OFF KING STREET TOWARDS GALVELMORE ST...     KING
STREET     FAST
7         8  51249.0            CAR PARK WEST END OFF ST NINIANS COURT   ATHOLL
STREET    RAPID
8         9  50279.0           CAR PARK OFF JUNCTION ROAD PARK AND RIDE   JUNCTIO
N ROAD     FAST
9        10  50280.0           CAR PARK OFF JUNCTION ROAD PARK AND RIDE   JUNCTIO
N ROAD     FAST

                     CH_TYPE      LOCATION            CON_TYPE CP_CHARGE  USAGE_
OUTPUT1_kW  OUTPUT2_kW  STAY  \
0          COMPACT TRI RAPID     ABERFELDY  TYPE 2;CHAdeMO;CCS      FREE  PUBLIC
50        43.0   1.0
1  STANDARD DUAL OUTLET POST     ABERFELDY        TYPE 2;TYPE2      FREE  PUBLIC
22.0   1.0
2  STANDARD DUAL OUTLET POST         ALYTH        TYPE 2;TYPE2      FREE  PUBLIC
22.0   1.0
3         QC45 RAPID CHARGER  AUCHTERARDER  TYPE 2;CHAdeMO;CCS      FREE  PUBLIC
50        43.0   1.0
4          COMPACT TRI RAPID   BLAIRGOWRIE  TYPE 2;CHAdeMO;CCS    CHARGE  PUBLIC
50        43.0   1.0
5          COMPACT TRI RAPID        CRIEFF  TYPE 2;CHAdeMO;CCS      FREE  PUBLIC
50        43.0   1.0
6  STANDARD DUAL OUTLET POST        CRIEFF        TYPE 2;TYPE2      FREE  PUBLIC
22.0   1.0
7         QC45 RAPID CHARGER       DUNKELD  TYPE 2;CHAdeMO;CCS    CHARGE  PUBLIC
50        43.0   1.0
8  STANDARD DUAL OUTLET POST       KINROSS        TYPE 2;TYPE2      FREE  PUBLIC
22.0   1.0
9  STANDARD DUAL OUTLET POST       KINROSS        TYPE 2;TYPE2      FREE  PUBLIC
22.0   1.0

             x           y
0  285847.9052  749128.0441
1  285851.3935  749129.9821
2  324594.7646  748530.0606
3  294480.8801  712789.2309
4  317925.2794  745163.0101
5  286263.6019  721608.8026
6  286267.1167  721607.7561
7  302613.3414  742878.7824
```

```
8  311211.8651  702495.0147
9  311214.4570  702495.4787

--- Dataset Info (Columns and Data Types) ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45 entries, 0 to 44
Data columns (total 15 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   MI_PRINX    45 non-null     int64
 1   Id          40 non-null     float64
 2   CP_Name     42 non-null     object
 3   ADDRESS     45 non-null     object
 4   CH_SPEED    45 non-null     object
 5   CH_TYPE     45 non-null     object
 6   LOCATION    45 non-null     object
 7   CON_TYPE    45 non-null     object
 8   CP_CHARGE   34 non-null     object
 9   USAGE_      45 non-null     object
 10  OUTPUT1_kW  36 non-null     object
 11  OUTPUT2_kW  44 non-null     float64
 12  STAY        34 non-null     float64
 13  x           45 non-null     float64
 14  y           45 non-null     float64
dtypes: float64(5), int64(1), object(9)
memory usage: 5.4+ KB
```

Cleaning the Location Data

As a further data cleaning step, I am removing any rows from the df_locations table that are missing a name for the charging point. The CP_Name is a critical piece of information for merging this dataset with my main session data, so any records without it cannot be used. I am also printing the total number of rows before and after this operation to keep a clear record of how many entries were removed.

In [ ]:
```python
# This code assumes 'df_locations' DataFrame from the previous step is in memory

# Remove rows with missing CP_Name ---

print("\n--- Checking for missing values in CP_Name ---")
# Store the number of rows before this cleaning step
rows_before_cp_name_clean = len(df_locations)

# Remove rows where the 'CP_Name' is missing.
df_locations.dropna(subset=['CP_Name'], inplace=True)

# Print the shape after cleaning
print(f"\nNumber of rows before removing missing CP_Name: {rows_before_cp_name_c
print(f"Number of rows after removing missing CP_Name: {len(df_locations)}")
print(f"Number of rows removed: {rows_before_cp_name_clean - len(df_locations)}"

print("\n Data cleaning for CP_Name complete.")
```

```
--- Checking for missing values in CP_Name ---

Number of rows before removing missing CP_Name: 45
Number of rows after removing missing CP_Name: 42
Number of rows removed: 3

✅ Data cleaning for CP_Name complete.
```

Map of EV Charger Types in Perth and Kinross

I create a folder called project_visualizations for the output, read the council's charger list from a CSV and the ward boundaries from a GeoJSON, rename the coordinate columns to latitude and longitude, coerce them to numbers, drop any rows without usable coordinates, and convert the British National Grid positions to standard GPS latitude and longitude so points land correctly on the map; I then label each site by speed using a simple rule of thumb where any entry whose CH_SPEED text contains "RAPID" is classed as Rapid and everything else becomes Fast or Standard, which seems a safer choice than guessing from inconsistent power labels; after that I load the boundary shapes and keep only Perth and Kinross, turn the charger table into a geo-enabled table, and draw a clear static map with the council outline in light grey, Rapid chargers shown as larger red stars, Fast or Standard chargers as smaller blue circles, a tidy legend, a title, and hidden axes; finally I save a high-resolution image called map_geographic_equity_by_speed.png in the project_visualizations folder and, if anything goes wrong or a required file is missing, I print a helpful message so the problem is obvious.

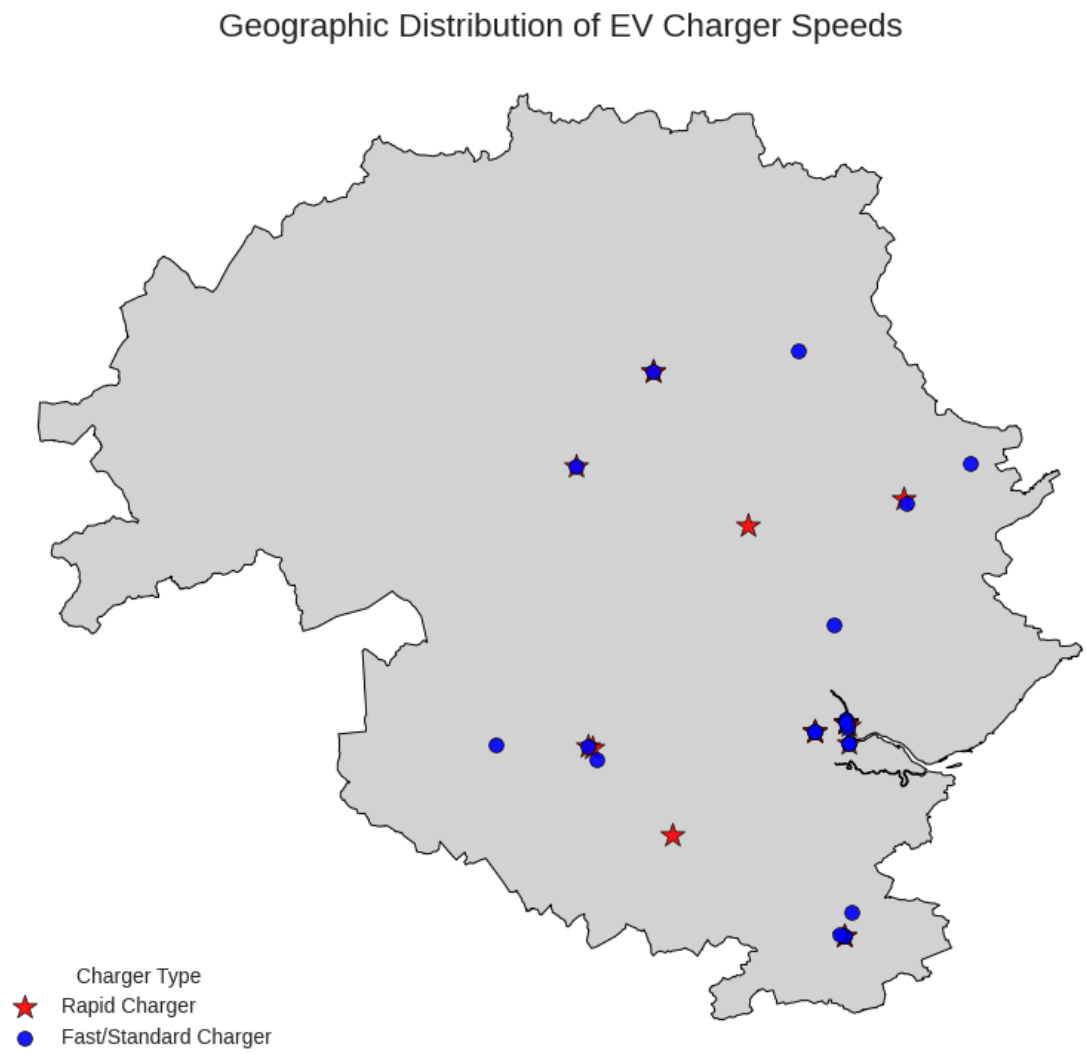
In [154…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import geopandas as gpd
from pyproj import Transformer

# setup

# Define the directory where the map will be saved
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)
print(f"The map will be saved to the '{SAVE_DIR}' directory.")

# --- File Paths ---
# This script assumes the necessary files are available.
LOCATIONS_FILE = "/content/drive/MyDrive/CIM/Dissertation/Perth and Kinross Coun
BOUNDARY_FILE = "ward_boundaries.geojson"

# Data preparation

def load_and_prepare_charger_data(locations_csv_path):
    """
    Loads and prepares the current list of all charger locations for mapping.
    """
    print("\n--- Preparing charger location data ---")
    try:
        # Load the official PKC charger locations
```

```
        df_locations = pd.read_csv(locations_csv_path)

        # --- Rename the coordinate columns and force numeric types ---
        df_locations.rename(columns={'x': 'longitude', 'y': 'latitude'}, inplace
        df_locations['latitude'] = pd.to_numeric(df_locations['latitude'], error
        df_locations['longitude'] = pd.to_numeric(df_locations['longitude'], err
        df_locations.dropna(subset=['latitude', 'longitude'], inplace=True)

        # --- Coordinate Transformation (British National Grid to WGS84) ---
        transformer = Transformer.from_crs("EPSG:27700", "EPSG:4326", always_xy=
        df_locations['lon_wgs84'], df_locations['lat_wgs84'] = transformer.trans
            df_locations['longitude'].values,
            df_locations['latitude'].values
        )

        # --- Categorize Charger Type based on Speed ---
        # This is the key step for this specific analysis.
        if 'CH_SPEED' in df_locations.columns:
            # We define 'Rapid' as anything with "RAPID" in the description, as
            df_locations['ChargerType'] = np.where(df_locations['CH_SPEED'].str.
                                                   'Rapid',
                                                   'Fast/Standard')
        else:
            df_locations['ChargerType'] = 'Unknown'

        print(f" Loaded and processed {len(df_locations)} charger locations.")
        return df_locations

    except Exception as e:
        print(f" Could not prepare charger data. Error: {e}")
        return None

# map generation

def create_distribution_map(charger_data, boundary_file_path, save_dir):
    """
    Generates and saves a static map showing the distribution of charger types.
    """
    print("\n--- Generating charger speed distribution map ---")
    try:
        # --- Load and prepare the boundary data ---
        gdf_boundaries = gpd.read_file(boundary_file_path)
        pkc_boundary = gdf_boundaries[gdf_boundaries['LAD13NM'] == 'Perth and Ki

        # --- Convert charger data to a GeoDataFrame ---
        gdf_chargers = gpd.GeoDataFrame(
            charger_data,
            geometry=gpd.points_from_xy(charger_data.lon_wgs84, charger_data.lat
            crs="EPSG:4326"
        )

        # --- Plotting ---
        plt.style.use('seaborn-v0_8-whitegrid')
        fig, ax = plt.subplots(1, 1, figsize=(10, 12))

        # Plot the PKC boundary as the base layer
        pkc_boundary.plot(ax=ax, color='lightgray', edgecolor='black', linewidth

        # Define markers and colors for different charger types
        type_aesthetics = {
```

```
            'Rapid': {'marker': '*', 'color': 'red', 's': 150, 'label': 'Rapid C
            'Fast/Standard': {'marker': 'o', 'color': 'blue', 's': 50, 'label':
        }

        # Plot each charger type separately to create a clear legend
        for charger_type, aesthetics in type_aesthetics.items():
            gdf_subset = gdf_chargers[gdf_chargers['ChargerType'] == charger_typ
            if not gdf_subset.empty:
                ax.scatter(
                    gdf_subset.geometry.x,
                    gdf_subset.geometry.y,
                    marker=aesthetics['marker'],
                    s=aesthetics['s'],
                    c=aesthetics['color'],
                    label=aesthetics['label'],
                    alpha=0.9,
                    edgecolors='k',
                    linewidth=0.5,
                    zorder=10
                )

        ax.set_title('Geographic Distribution of EV Charger Speeds', fontsize=16
        ax.set_axis_off() # Hide the lat/lon axes for a clean map
        ax.legend(title="Charger Type", loc='lower left', fancybox=True, shadow=

        # Save the map to a file
        map_path = os.path.join(save_dir, "map_geographic_equity_by_speed.png")
        plt.savefig(map_path, dpi=1200, bbox_inches='tight')
        plt.show()
        plt.close()

        print(f"\n Distribution map saved successfully to: {map_path}")

    except Exception as e:
        print(f" Could not create the distribution map. Error: {e}")

# Main execution

if __name__ == "__main__":
    if os.path.exists(LOCATIONS_FILE) and os.path.exists(BOUNDARY_FILE):
        # Run the full pipeline
        charger_data = load_and_prepare_charger_data(LOCATIONS_FILE)
        if charger_data is not None:
            create_distribution_map(charger_data, BOUNDARY_FILE, SAVE_DIR)
    else:
        print(f" ERROR: Required file not found. Please ensure both '{LOCATIONS_
```

```
The map will be saved to the 'project_visualizations' directory.

--- Preparing charger location data ---
 Loaded and processed 45 charger locations.

--- Generating charger speed distribution map ---
```




```
Distribution map saved successfully to: project_visualizations/map_geographic_eq
uity_by_speed.png
```

Overall Interpretation The map reveals a "hub and spoke" strategy for infrastructure deployment. There is a clear concentration of chargers in the urban center of Perth, with "spokes" of infrastructure extending out to serve key towns and tourist routes. While this provides good coverage for major population centers and travel corridors, it also highlights potential gaps in service for more remote, rural areas.

Key Observations and Discussion Points: Urban Concentration: The densest cluster of both Rapid (red stars) and Fast/Standard (blue circles) chargers is located in and around the city of Perth in the south-east of the region. This is logical, as it serves the largest population center and the highest traffic areas.

Strategic Placement of Rapid Chargers: The red stars are not just in Perth. They are strategically placed in key towns along what are likely major A-roads (e.g., Auchterarder, Crieff, Dunkeld, Pitlochry).

Socio-Technical Insight: This is a deliberate policy to support longer-distance travel and tourism. It ensures that drivers can travel through the region with confidence, knowing they can get a quick top-up in major towns. This is a positive finding regarding enabling EV use beyond local commuting.

The Rural "Charging Desert" Question: While there is coverage in smaller towns, the vast northern and western parts of the region have very sparse infrastructure.

Socio-Technical Insight: This is the core of your geographic equity analysis. A resident living in the north-western part of the council area has significantly poorer access to public charging than someone in Perth. This could be a major barrier to EV adoption for rural communities, creating a two-tiered system where EVs are practical for urban dwellers but not for those in more remote areas.

How to Use This in Your Dissertation: Argument: You can argue that while the council has successfully supported major transport routes, a significant geographic disparity remains.

Evidence: Use this map as "Figure X" to visually prove your point. You can state, "As shown in Figure X, the distribution of high-speed Rapid chargers is primarily concentrated along the A9 corridor and within the Perth urban area, leaving large rural expanses underserved."

Recommendation: This leads directly to a policy recommendation. For example: "To ensure equitable access and encourage rural EV adoption, a future phase of infrastructure deployment should focus on placing at least one Rapid or Fast charger in key villages in the currently underserved north-western region."

Economic Accessibility Chart for EV Charging Sites

I set up an output folder called project_visualizations, read the official charger list from the CSV, and tidy the data before plotting; first I drop entries with a missing charger name, rename CP_Name to Site, trim stray spaces, and standardise a handful of awkward site labels so locations that are really the same place show up under one clear name; I then create a simple payment flag by reading CP_CHARGE, mapping FREE to Free and CHARGE to Paid, and marking anything unclear as Unknown, which feels safer than guessing; next I build a table that counts, for each site, how many chargers are Free, Paid, or Unknown, add a total column, and keep the 15 busiest sites so the figure stays readable; after sorting by that total, I drop the helper column and draw a horizontal stacked bar chart where each bar shows the mix of free and paid sockets at a site, add a title, axis labels, a neat legend, and light gridlines, and save a high-resolution image called economic_accessibility_by_site.png in the project_visualizations folder; along the way I print friendly status messages so it is obvious what ran, and I leave the plotting function without a try–except so, if something breaks, I see the full error and can fix it quickly.

In [155…

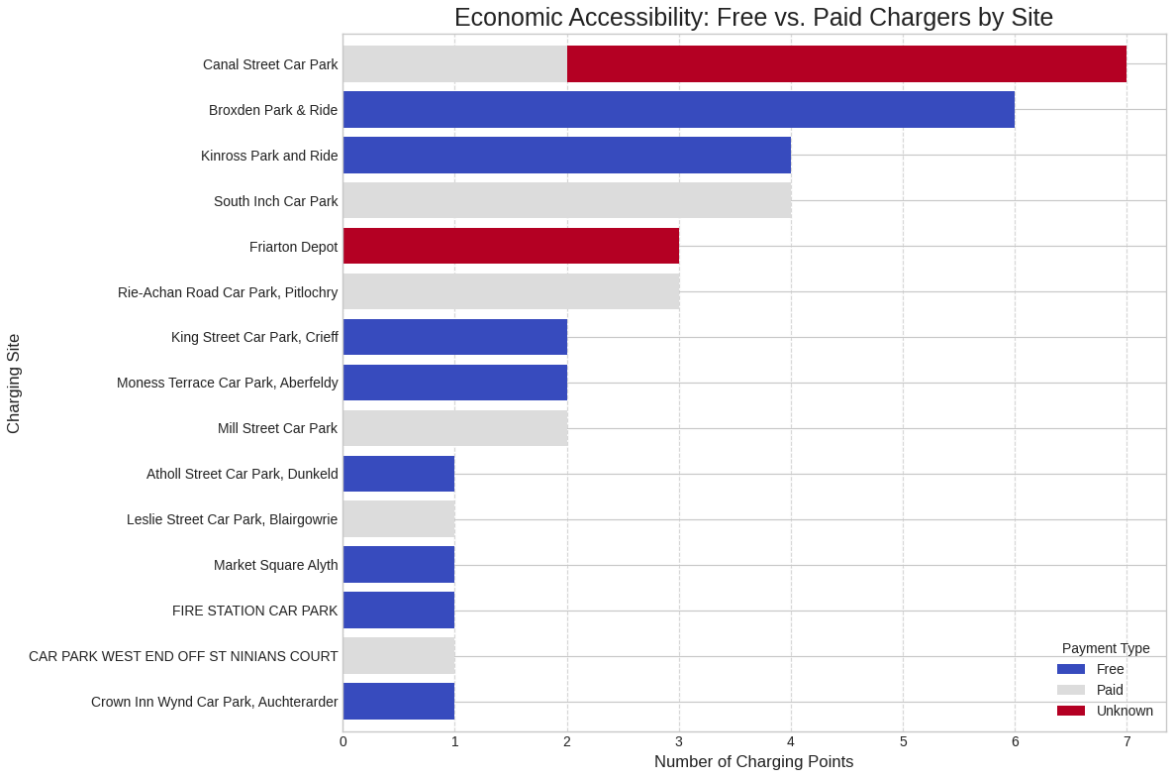
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Setup
```

```
# Define the directory where the plot will be saved
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)
print(f"The analysis chart will be saved to the '{SAVE_DIR}' directory.")

# --- File Path ---
LOCATIONS_FILE = "/content/drive/MyDrive/CIM/Dissertation/Perth and Kinross Coun

# Data preparation

def prepare_economic_analysis_data(locations_csv_path):
    """
    Loads and prepares the current list of all charger locations for economic an
    """
    print("\n--- Preparing data for economic accessibility analysis ---")
    try:
        # Load the official PKC charger locations
        df_locations = pd.read_csv(locations_csv_path)

        # --- NEW: Remove rows where the charger name is missing before any proc
        df_locations.dropna(subset=['CP_Name'], inplace=True)

        # --- Standardize column names ---
        df_locations.rename(columns={'CP_Name': 'Site'}, inplace=True)
        df_locations['Site'] = df_locations['Site'].astype(str).str.strip()

        # --- Clean and standardize site names ---
        location_name_map = {
            'ACCESS ROAD INTO PARK AND RIDE BROXDEN AVENUE': 'Broxden Park & Rid
            'CAR PARK OFF JUNCTION ROAD PARK AND RIDE': 'Kinross Park and Ride',
            'CAR PARK OFF SHORE ROAD AT SOUTH INCH': 'South Inch Car Park',
            'CANAL STREETMULTY STOREY CAR PARK': 'Canal Street Car Park',
            'CAR PARK MULTI STOREY OFF CANAL STREET': 'Canal Street Car Park',
            'CAR PARK OFF KING STREET TOWARDS GALVELMORE STREET': 'King Street C
            'CAR PARK OFF HIGH STREET': 'Crown Inn Wynd Car Park, Auchterarder',
            'CAR PARK OFF CROFT LANE AT LESLIE STREET': 'Leslie Street Car Park,
            'CAR PARK OFF RIE-ACHAN ROAD': 'Rie-Achan Road Car Park, Pitlochry',
            'CAR PARK OFF MILL STREET EAST': 'Mill Street Car Park',
            'CAR PARK OFF MONESS TERRACE': 'Moness Terrace Car Park, Aberfeldy',
            'FRIARTON WASTE TRANSFER STATION': 'Friarton Depot',
            'CAR PARK IN MARKET SQUARE': 'Market Square Alyth',
            'LEADENFLOWER CAR PARK': 'Atholl Street Car Park, Dunkeld'
        }
        df_locations['Site'] = df_locations['Site'].replace(location_name_map)

        # --- Use the 'CP_CHARGE' column for payment type ---
        if 'CP_CHARGE' in df_locations.columns:
            df_locations['PaymentType'] = df_locations['CP_CHARGE'].str.strip().
                'FREE': 'Free',
                'CHARGE': 'Paid'
            }).fillna('Unknown') # Handle any missing payment info
        else:
            df_locations['PaymentType'] = 'Unknown'

        print(f" Successfully loaded and processed data for {len(df_locations)}
        return df_locations

    except Exception as e:
```

```
        print(f" Could not prepare analysis data. Error: {e}")
        return None

# Bar chart generation

def create_economic_accessibility_chart(df_locations, save_dir):
    """
    Generates and saves a stacked bar chart showing the distribution of
    free vs. paid chargers at each major site.
    """
    print("\n--- Generating economic accessibility chart ---")
    # Removed the try-except block as requested to show the full traceback on er

    # Create a cross-tabulation of site vs. payment type
    crosstab = pd.crosstab(df_locations['Site'], df_locations['PaymentType'])

    # We only want to plot sites with a significant number of chargers
    crosstab['total'] = crosstab.sum(axis=1)
    top_sites_crosstab = crosstab.sort_values('total', ascending=False).head(15)

    # --- CORRECTED: Sort by the total first, then drop the helper column ---
    # This ensures the plot is ordered correctly from busiest to least busy site
    plot_data = top_sites_crosstab.sort_values(by='total', ascending=True)
    plot_data = plot_data.drop(columns='total')

    # Plotting
    plt.style.use('seaborn-v0_8-whitegrid')
    ax = plot_data.plot(
        kind='barh',
        stacked=True,
        figsize=(12, 8),
        colormap='coolwarm',
        width=0.8
    )

    ax.set_title('Economic Accessibility: Free vs. Paid Chargers by Site', fonts
    ax.set_xlabel('Number of Charging Points', fontsize=12)
    ax.set_ylabel('Charging Site', fontsize=12)
    ax.legend(title='Payment Type')
    ax.grid(axis='x', linestyle='--', alpha=0.7)

    plt.tight_layout()

    # Save the figure
    save_path = os.path.join(save_dir, "economic_accessibility_by_site.png")
    plt.savefig(save_path, dpi=1200)
    plt.show()
    plt.close()

    print(f"\n Economic accessibility chart saved successfully to: {save_path}")


# Main execution

if __name__ == "__main__":
    if os.path.exists(LOCATIONS_FILE):
        # Run the full pipeline
        df_locations = prepare_economic_analysis_data(LOCATIONS_FILE)
        if df_locations is not None:
            create_economic_accessibility_chart(df_locations, SAVE_DIR)
```

```
    else:
        print(f" ERROR: The locations file was not found at '{LOCATIONS_FILE}'.")
```

```
The analysis chart will be saved to the 'project_visualizations' directory.

--- Preparing data for economic accessibility analysis ---
 Successfully loaded and processed data for 42 charging points.

--- Generating economic accessibility chart ---
```




```
 Economic accessibility chart saved successfully to: project_visualizations/econo
mic_accessibility_by_site.png
```

Overall Interpretation The most striking takeaway is that the public charging network in this dataset is overwhelmingly free to use. The vast majority of charging points, represented by the blue bars, do not require payment. This indicates a deliberate strategy by the council to encourage EV adoption by removing one of the primary cost barriers.

However, the distribution of these free chargers is not uniform, which leads to some interesting social and economic questions.

Breakdown by Key Sites: Major Transport Hubs (Broxden & Kinross Park & Ride):

What it shows: These are two of the largest sites, and they are composed almost entirely of free chargers.

Interpretation for your dissertation: This is a strong policy signal. The council is actively incentivizing commuters to use Park & Ride facilities by offering free charging. This is a strategy to reduce city-center congestion and emissions. You can discuss this as a successful example of integrated transport and energy policy.

City Centre Hubs (Canal Street & South Inch Car Parks):

What it shows: These sites show a more complex picture. While they have many free chargers, they also have a significant number of "Paid" or "Unknown" chargers. Canal Street, the largest single site, has the most diverse mix.

Interpretation for your dissertation: This is where your socio-technical analysis becomes crucial. Why is there a mix of payment types in the city centre? It could be that the free chargers are lower power (Fast/Standard) while the paid ones are high-power (Rapid). Or it could be a strategy to manage demand in high-traffic areas. This "hybrid" model at key urban locations is a key finding.

Specialized Sites (Friarton Depot):

What it shows: This site is composed entirely of chargers with an "Unknown" payment type.

Interpretation for your dissertation: Given that this is a depot, these are likely fleet chargers not intended for the general public. The "Unknown" status might mean they are privately managed or have a different internal billing system. This highlights that not all "public" infrastructure serves the same purpose or user group.

Key Insights for Your Dissertation: A "Freemium" Model: The network operates on a "freemium" model. Basic access is largely free, especially at strategic locations like Park & Rides, to encourage adoption. However, in the most high-demand urban locations, a paid or mixed model exists.

Data Quality as a Finding: The presence of the "Unknown" category is itself a finding. It points to inconsistencies in the public data, which is a real-world challenge for any analysis. You can mention this as a limitation and a recommendation for improvement in data collection practices.

The Question of Equity: This plot allows you to ask critical questions. Does providing free charging at Park & Ride facilities primarily benefit affluent commuters with EVs? Are there enough free chargers in residential areas for people who don't have private driveways? This visualization is the perfect starting point for that discussion.

Creating a Clean, Enriched Dataset of Charger Locations

I read the council’s charger list from the CSV, drop entries that lack a name or coordinates, and rename the key columns so CP_Name becomes Site and x and y become longitude and latitude; I tidy the Site text and standardise several awkward labels so places that are really the same show up under one clear name, which should make later analysis less noisy; I add simple features that I will reuse throughout the project, including ChargerType, where any CH_SPEED text containing “RAPID” is marked Rapid and everything else becomes Fast or Standard, PaymentType, where CP_CHARGE is mapped to Free or Paid with unknowns left as Unknown, and LocationType, which classifies each Site as a transport hub, depot, public car park, or on-street or other based on the wording; I also create MaxStayHrs by turning the STAY field into a number and, if

that is missing, assuming 24 hours so I have a sensible ceiling rather than blanks; once the table is prepared I print a short success message and return it, then in the main block I only run the pipeline if the file exists and show a quick preview of the cleaned data, which helps me confirm the transformations look plausible before I use the dataset in plots or models.

In [156…

```
import pandas as pd
import numpy as np
import os

# setup
# This script assumes the locations CSV file is available at the specified path.

LOCATIONS_FILE = "/content/drive/MyDrive/CIM/Dissertation/Perth and Kinross Coun

# Data preparation

def create_prepared_locations_dataframe(locations_csv_path):
    """
    Loads and prepares the current list of all charger locations, returning a
    single, clean DataFrame for all subsequent analyses.
    """
    print("\n--- Preparing Final Analysis DataFrame ---")
    try:
        # Load the official PKC charger locations
        df = pd.read_csv(locations_csv_path)

        # --- a. Handle Missing Data ---
        # Remove rows where the charger name or coordinates are missing
        df.dropna(subset=['CP_Name', 'x', 'y'], inplace=True)

        # --- b. Standardize Column Names ---
        df.rename(columns={'CP_Name': 'Site', 'x': 'longitude', 'y': 'latitude'}

        # --- c. Clean and Standardize Site Names ---
        df['Site'] = df['Site'].astype(str).str.strip()
        location_name_map = {
            'ACCESS ROAD INTO PARK AND RIDE BROXDEN AVENUE': 'Broxden Park & Rid
            'CAR PARK OFF JUNCTION ROAD PARK AND RIDE': 'Kinross Park and Ride',
            'CAR PARK OFF SHORE ROAD AT SOUTH INCH': 'South Inch Car Park',
            'CANAL STREETMULTY STOREY CAR PARK': 'Canal Street Car Park',
            'CAR PARK MULTI STOREY OFF CANAL STREET': 'Canal Street Car Park',
            'CAR PARK OFF KING STREET TOWARDS GALVELMORE STREET': 'King Street C
            'CAR PARK OFF HIGH STREET': 'Crown Inn Wynd Car Park, Auchterarder',
            'CAR PARK OFF CROFT LANE AT LESLIE STREET': 'Leslie Street Car Park,
            'CAR PARK OFF RIE-ACHAN ROAD': 'Rie-Achan Road Car Park, Pitlochry',
            'CAR PARK OFF MILL STREET EAST': 'Mill Street Car Park',
            'CAR PARK OFF MONESS TERRACE': 'Moness Terrace Car Park, Aberfeldy',
            'FRIARTON WASTE TRANSFER STATION': 'Friarton Depot',
            'CAR PARK IN MARKET SQUARE': 'Market Square Alyth',
            'LEADENFLOWER CAR PARK': 'Atholl Street Car Park, Dunkeld'
        }
        df['Site'] = df['Site'].replace(location_name_map)

        # --- d. Create Feature: ChargerType ---
        if 'CH_SPEED' in df.columns:
            df['ChargerType'] = np.where(df['CH_SPEED'].str.contains("RAPID", ca
        else:
```

```
            df['ChargerType'] = 'Unknown'

        # --- e. Create Feature: PaymentType ---
        if 'CP_CHARGE' in df.columns:
            df['PaymentType'] = df['CP_CHARGE'].str.strip().replace({'FREE': 'Fr
        else:
            df['PaymentType'] = 'Unknown'

        # --- f. Create Feature: LocationType ---
        def get_location_type(site_name):
            if "Park & Ride" in site_name: return "Transport Hub (Park & Ride)"
            if "Depot" in site_name: return "Depot / Fleet"
            if "Car Park" in site_name: return "Public Car Park"
            return "On-Street / Other"
        df['LocationType'] = df['Site'].apply(get_location_type)

        # --- g. Create Feature: MaxStayHrs ---
        if 'STAY' in df.columns:
            df['MaxStayHrs'] = pd.to_numeric(df['STAY'], errors='coerce').fillna
        else:
            df['MaxStayHrs'] = 24 # Default to 24h if column is missing

        print(f" Successfully created the final prepared DataFrame with {len(df)
        return df

    except Exception as e:
        print(f" Could not prepare analysis data. Error: {e}")
        return None

# Main Execution

if __name__ == "__main__":
    if os.path.exists(LOCATIONS_FILE):
        # Create the final, clean DataFrame
        df_locations_prepared = create_prepared_locations_dataframe(LOCATIONS_FI
        if df_locations_prepared is not None:
            print("\n--- Preview of Prepared Data ---")
            print(df_locations_prepared.head())
    else:
        print(f" ERROR: The locations file was not found at '{LOCATIONS_FILE}'."
```

```
--- Preparing Final Analysis DataFrame ---
 Successfully created the final prepared DataFrame with 42 charging points.

--- Preview of Prepared Data ---
   MI_PRINX       Id                                    Site         ADDRESS CH_SP
EED                   CH_TYPE  \
0         1  50995.0     Moness Terrace Car Park, Aberfeldy  MONESS TERRACE    RA
PID          COMPACT TRI RAPID
1         2  50811.0     Moness Terrace Car Park, Aberfeldy  MONESS TERRACE     F
AST  STANDARD DUAL OUTLET POST
2         3  51261.0                   Market Square Alyth   MARKET SQUARE     F
AST  STANDARD DUAL OUTLET POST
3         4  51250.0  Crown Inn Wynd Car Park, Auchterarder  CROWN INN WYND    RA
PID         QC45 RAPID CHARGER
4         5  50994.0    Leslie Street Car Park, Blairgowrie   LESLIE STREET    RA
PID          COMPACT TRI RAPID

       LOCATION            CON_TYPE CP_CHARGE  USAGE_ OUTPUT1_kW  OUTPUT2_kW  STA
Y    longitude     latitude  \
0     ABERFELDY  TYPE 2;CHAdeMO;CCS      FREE  PUBLIC         50        43.0   1.
0  285847.9052  749128.0441
1     ABERFELDY        TYPE 2;TYPE2      FREE  PUBLIC                   22.0   1.
0  285851.3935  749129.9821
2         ALYTH        TYPE 2;TYPE2      FREE  PUBLIC                   22.0   1.
0  324594.7646  748530.0606
3  AUCHTERARDER  TYPE 2;CHAdeMO;CCS      FREE  PUBLIC         50        43.0   1.
0  294480.8801  712789.2309
4   BLAIRGOWRIE  TYPE 2;CHAdeMO;CCS    CHARGE  PUBLIC         50        43.0   1.
0  317925.2794  745163.0101

     ChargerType PaymentType       LocationType  MaxStayHrs
0          Rapid        Free    Public Car Park         1.0
1  Fast/Standard        Free    Public Car Park         1.0
2  Fast/Standard        Free  On-Street / Other         1.0
3          Rapid        Free    Public Car Park         1.0
4          Rapid        Paid    Public Car Park         1.0
```

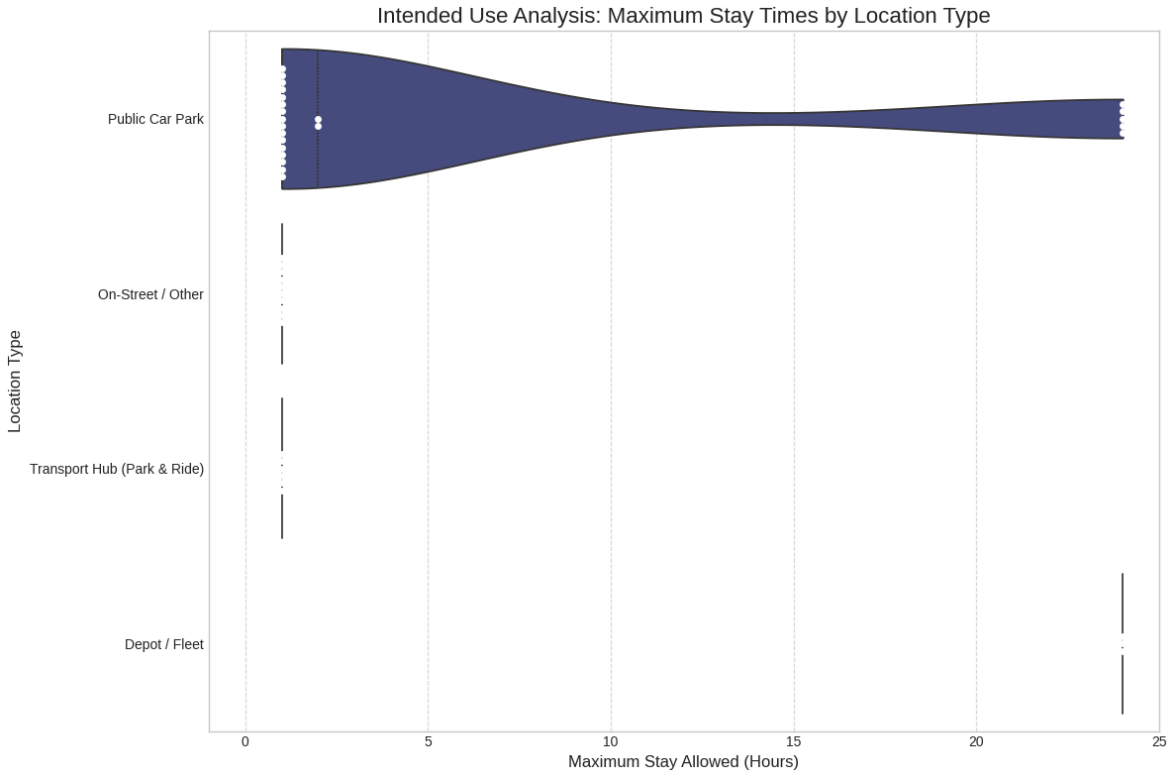
Intended Use Plot: Maximum Stay by Location Type

I assume the cleaned DataFrame is already in memory, set up an output folder called project_visualizations, and draw a single, readable chart that shows how maximum stay limits vary by place; I use a horizontal violin plot for each LocationType so the shape captures the full spread of stay times, add the inner quartile marks to hint at the middle range, and layer a light swarm of individual points on top to keep the real observations visible rather than hiding everything behind summaries; I stick with a simple white-grid style, label the axes, add a clear title, and cap the x axis at about a day, from just below zero to 25 hours, which keeps attention on the common short-stay policies while still catching anything near 24; once drawn, I save a high-resolution image named intended_use_by_stay_time.png inside project_visualizations and print a short success message, and if the DataFrame is missing or something else trips an error I report that plainly so I can fix it quickly.

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import os

# Setup
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)

# This script assumes the 'df_locations_prepared' DataFrame is already in memory

# Plot generation
print("\n--- Generating Intended Use (Max Stay) plot ---")
try:
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(12, 8))

    # --- IMPROVED VISUALIZATION: Use a Violin Plot with Jitter ---
    # A violin plot is better for showing the distribution of points.
    # The inner 'quartile' shows the box plot ranges, and the swarm shows indivi
    sns.violinplot(
        data=df_locations_prepared,
        x='MaxStayHrs',
        y='LocationType',
        orient='h',
        palette='viridis',
        inner='quartile', # Shows the quartiles inside the violin
        cut=0 # Prevents the violin from extending beyond the data range
    )

    # Overlay a swarm plot to show individual data points (jitter)
    sns.swarmplot(
        data=df_locations_prepared,
        x='MaxStayHrs',
        y='LocationType',
        orient='h',
        color='white',
        edgecolor='gray',
        size=5
    )

    plt.title('Intended Use Analysis: Maximum Stay Times by Location Type', font
    plt.xlabel('Maximum Stay Allowed (Hours)', fontsize=12)
    plt.ylabel('Location Type', fontsize=12)
    plt.grid(axis='x', linestyle='--', alpha=0.7)

    # Set a more reasonable x-axis limit to focus on the main distribution
    # We can cap it slightly above the most common short-stay limits.
    plt.xlim(-1, 25)

    plt.tight_layout()

    # Save the figure
    save_path = os.path.join(SAVE_DIR, "intended_use_by_stay_time.png")
    plt.savefig(save_path, dpi=1200)
    plt.show()
    plt.close()

    print(f"\n Intended use chart saved successfully to: {save_path}")
```

```
except NameError:
    print(" ERROR: The 'df_locations_prepared' DataFrame was not found.")
    print("Please ensure you have successfully run the 'Final Data Preparation f
except Exception as e:
    print(f" Could not create the analysis chart. Error: {e}")
```

```
--- Generating Intended Use (Max Stay) plot ---
```




```
✅ Intended use chart saved successfully to: project_visualizations/intended_use
_by_stay_time.png
```

Overall Interpretation The plot reveals a clear and deliberate strategy by the council to design different types of charging infrastructure for different social purposes. There is a strong distinction between infrastructure designed for short-term, transient use (like public car parks) and infrastructure for long-duration, depot-based charging.

Breakdown by Location Type: Public Car Park:

What it shows: This is the most interesting category. The "violin" is very wide and heavily concentrated between 1 and 4 hours. The white dots (representing individual chargers) are all clustered here.

Interpretation: This indicates that the vast majority of chargers in public car parks are explicitly designed for short-stay users, such as shoppers or visitors. The goal is to encourage quick turnover, not all-day parking. The single outlier point at 24 hours represents a long-stay car park, but it is the exception, not the rule.

Transport Hub (Park & Ride) and On-Street / Other:

What it shows: Both of these categories have very thin violins, with all data points clustered at very short stay times (around 1-2 hours).

Interpretation: This is a very significant finding. It suggests that even at Park & Ride facilities, which are typically used by all-day commuters, the chargers are intended for rapid top-ups, not for commuters to leave their cars charging all day. This highlights a potential mismatch between infrastructure design and user needs.

Depot / Fleet:

What it shows: The data points are all clustered at the 24-hour mark.

Interpretation: This makes perfect sense. These chargers are not for the general public but for council vehicles or other fleets that are parked overnight. The 24-hour limit essentially means there is "no limit," allowing for slow, full charging.

Key Insights for Your Dissertation: Focus on Transient Charging: The public-facing network is overwhelmingly designed to support transient charging (short trips, shopping) rather than solving the problem for residents without private driveways or all-day commuters.

The Commuter Problem: The short stay limits at Park & Ride hubs are a key socio-technical finding. You can argue that this infrastructure design does not currently serve the needs of commuters who might want to leave their car for 8-9 hours. This could be a barrier to EV adoption for this group.

Clear Policy Segmentation: The plot provides strong evidence that the council has different policies for different locations. Your dissertation can discuss the pros and cons of this approach—it ensures high turnover in busy car parks but may leave certain user groups (like commuters) underserved.

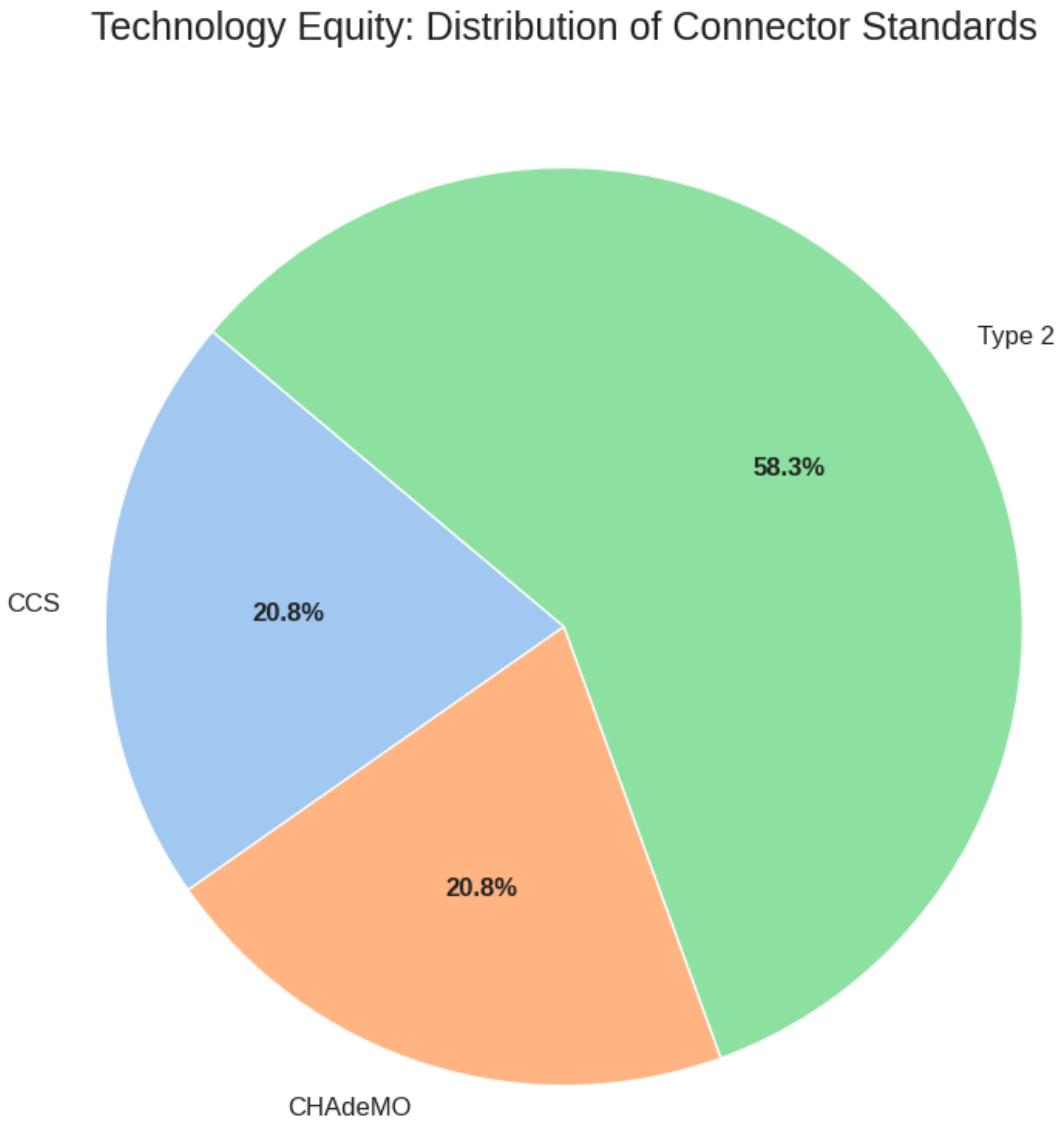
Technology Equity: Connector Standards Pie Chart

I assume the cleaned DataFrame is already in memory, create an output folder if needed, then scan the CON_TYPE text for three common plug standards, CCS, CHAdeMO and Type 2, counting how many chargers mention each one regardless of letter case; because a single charger can list more than one connector, these counts may overlap, which is fine for this purpose since I just want a sense of the mix on the ground; I turn the counts into a small table and draw a simple pie chart with percentage labels, a soft colour palette, a clear title and tidy white edges on each slice so the segments are easy to read; finally I save a high-resolution image called technology_equity_connector_types.png in the project_visualizations folder and print a short success message, and if the prepared DataFrame is missing or something else fails I show a plain error so I know what to fix.

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import os
import pandas as pd

# Setup
SAVE_DIR = "project_visualizations"
```

```
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)

# This script assumes the 'df_locations_prepared' DataFrame is already in memory

# Plot generation
print("\n--- Generating Technology Equity (Connector Type) plot ---")
try:
    # --- Data Analysis ---
    # The CON_TYPE column often contains multiple values separated by semicolons
    # We need to count the presence of each key standard.
    # Note: A single charger can have multiple connectors.

    connector_counts = {
        'CCS': df_locations_prepared['CON_TYPE'].str.contains('CCS', case=False)
        'CHAdeMO': df_locations_prepared['CON_TYPE'].str.contains('CHAdeMO', cas
        'Type 2': df_locations_prepared['CON_TYPE'].str.contains('TYPE 2', case=
    }

    # Convert to a pandas Series for easy plotting
    connector_series = pd.Series(connector_counts)

    # --- Visualization ---
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(10, 8))

    # Create the pie chart
    wedges, texts, autotexts = plt.pie(
        connector_series,
        labels=connector_series.index,
        autopct='%1.1f%%',
        startangle=140,
        colors=sns.color_palette('pastel'),
        wedgeprops={'edgecolor': 'white', 'linewidth': 1}
    )

    # Improve text formatting
    plt.setp(autotexts, size=12, weight="bold")
    plt.setp(texts, size=12)

    plt.title('Technology Equity: Distribution of Connector Standards', fontsize
    plt.ylabel('') # Hide the y-label for a pie chart

    plt.tight_layout()

    # Save the figure
    save_path = os.path.join(SAVE_DIR, "technology_equity_connector_types.png")
    plt.savefig(save_path, dpi=1200)
    plt.show()
    plt.close()

    print(f"\n Connector type distribution chart saved successfully to: {save_pa

except NameError:
    print(" ERROR: The 'df_locations_prepared' DataFrame was not found.")
    print("Please ensure you have successfully run the 'Final Data Preparation f
except Exception as e:
    print(f" Could not create the analysis chart. Error: {e}")
```

```
--- Generating Technology Equity (Connector Type) plot ---
```



✅ Connector type distribution chart saved successfully to: project_visualizations/technology_equity_connector_types.png

Overall Interpretation The chart reveals a network that was well-designed for the EV market of the past but now faces a significant challenge in serving the market of the future. While it provides excellent coverage for all types of connectors currently on the road, the heavy investment in the older CHAdeMO standard creates a potential "technology bottleneck" for the growing number of modern EV drivers.

Breakdown by Connector Type: Type 2 (58.3%): The Workhorse Connector

What it is: This is the universal standard for AC "Fast/Standard" charging across the UK and Europe. It's the most common plug type on the network.

Interpretation: This is a positive finding. It means that almost all EV drivers can use the majority of the charging sockets for slower, longer-duration charging (e.g., while parked for a few hours). This ensures broad basic compatibility.

CCS and CHAdeMO (20.8% each): The Rapid Charging Dilemma

What they are: These are the two competing standards for DC Rapid charging, which is essential for long journeys and quick top-ups.

CHAdeMO: The older standard, primarily used by the Nissan Leaf (one of the most popular early EVs) and some other Japanese models.

CCS: The modern standard used by virtually all new European, Korean, and American EVs (e.g., Tesla, VW, Hyundai, Kia, Ford).

Interpretation: The almost perfect 50/50 split between CCS and CHAdeMO is the most critical finding. It indicates that the network was built out during a time when the Nissan Leaf was a dominant force in the EV market. The council made a commendable effort to provide equal access for both types of vehicles.

Key Insights for Your Dissertation: Legacy Support vs. Future Demand: The network is very equitable for current and past EV owners. However, the UK car market has decisively moved to the CCS standard. This means that as older CHAdeMO vehicles are retired, half of the rapid charging infrastructure will serve a shrinking fraction of the EV fleet.

The Future Bottleneck: This creates a significant "technology equity" issue for the future. A driver of a new Volkswagen ID.4 (which uses CCS) effectively sees only half of the available rapid chargers as usable. This can lead to queues and frustration, even if the CHAdeMO plug right next to it is empty.

Policy Recommendation: This analysis provides a strong, data-driven basis for a policy recommendation. You can argue that any future investment should prioritize adding more CCS connectors. This could involve either building new sites with a CCS-first approach or retrofitting existing rapid chargers to have more CCS plugs, ensuring the public investment remains relevant and serves the majority of future EV drivers.

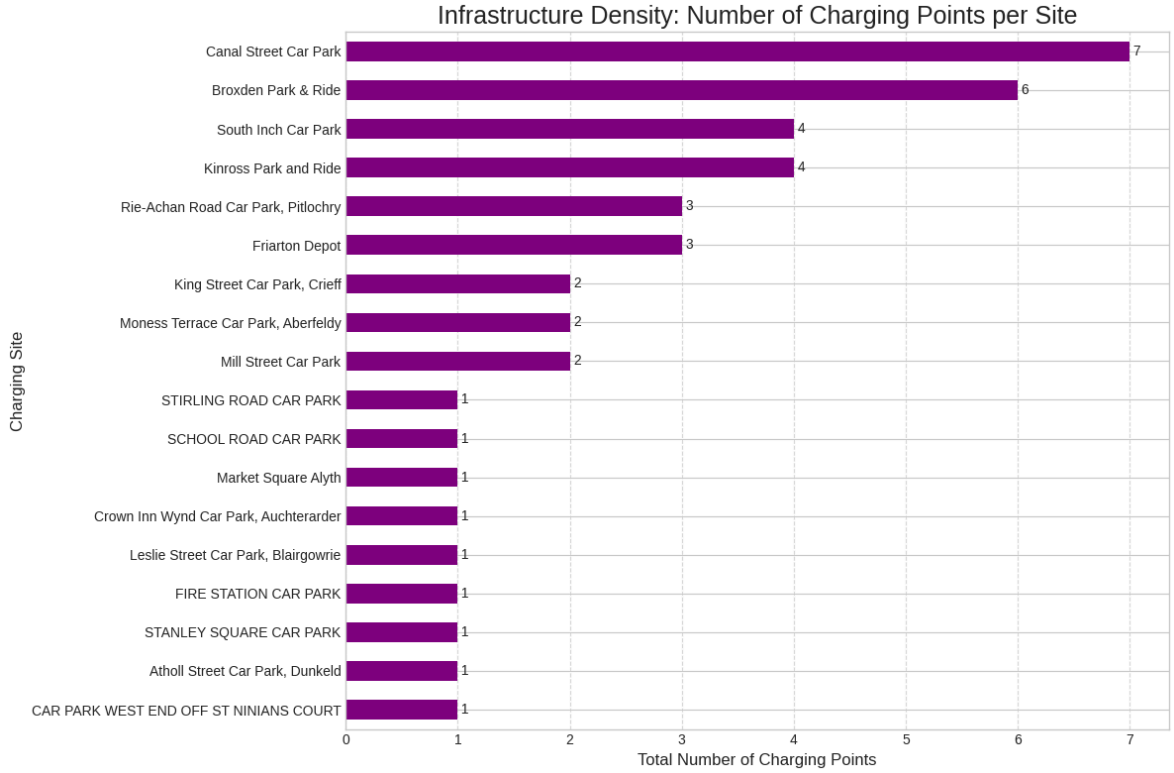
Infrastructure Density: Hubs vs Spokes

I assume the cleaned DataFrame is already in memory, create the project_visualizations folder if it does not exist, then count how many charging points appear at each Site so I can see where capacity clusters; I sort those counts for readability and draw a horizontal bar chart with a simple white grid, add a clear title and axis labels, and place small number labels at the end of each bar so the totals are obvious without squinting; the picture helps me spot hubs with many sockets versus smaller spoke sites with only one or two, which is useful when thinking about resilience and queuing; once the figure looks tidy, I save a high resolution image called infrastructure_density_hubs_vs_spokes.png in the project_visualizations folder and print a short success message, and if the DataFrame is missing or anything else goes wrong I surface a plain error so I know what to fix.

In [ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import os
import pandas as pd

# Setup
```

```
SAVE_DIR = "project_visualizations"
if not os.path.exists(SAVE_DIR):
    os.makedirs(SAVE_DIR)

# This script assumes the 'df_locations_prepared' DataFrame is already in memory

# Plot generation
print("\n--- Generating Infrastructure Density (Hubs vs. Spokes) plot ---")
try:
    # --- Data Analysis ---
    # Count the number of occurrences of each unique site name.
    # This gives us the number of charging points per site.
    site_counts = df_locations_prepared['Site'].value_counts()

    # --- Visualization ---
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(12, 8))

    # Create the horizontal bar chart, sorting the values for a clean look.
    site_counts.sort_values().plot(kind='barh', color='purple')

    plt.title('Infrastructure Density: Number of Charging Points per Site', font
    plt.xlabel('Total Number of Charging Points', fontsize=12)
    plt.ylabel('Charging Site', fontsize=12)
    plt.grid(axis='x', linestyle='--', alpha=0.7)

    # Add data labels to the end of each bar for clarity
    for index, value in enumerate(site_counts.sort_values()):
        plt.text(value, index, f' {value}', va='center')

    plt.tight_layout()

    # Save the figure
    save_path = os.path.join(SAVE_DIR, "infrastructure_density_hubs_vs_spokes.pn
    plt.savefig(save_path, dpi=1200)
    plt.show()
    plt.close()

    print(f"\n Infrastructure density chart saved successfully to: {save_path}")

except NameError:
    print(" ERROR: The 'df_locations_prepared' DataFrame was not found.")
    print("Please ensure you have successfully run the 'Final Data Preparation f
except Exception as e:
    print(f" Could not create the analysis chart. Error: {e}")
```

```
--- Generating Infrastructure Density (Hubs vs. Spokes) plot ---
```



✅ `Infrastructure density chart saved successfully to: project_visualizations/infrastructure_density_hubs_vs_spokes.png`

Overall Interpretation The chart clearly demonstrates a "Hub and Spoke" deployment strategy. The council has focused on creating a small number of large, multi-charger "Hubs" at key strategic locations, while simultaneously deploying a large number of single-charger "Spokes" to maximize geographic coverage.

Breakdown of the Strategy: The Hubs (High-Density Sites):

What it shows: You can see a few sites at the top of the chart with a significantly higher number of charging points than the rest. Canal Street Car Park (7 points), Broxden Park & Ride (6 points), and South Inch/Kinross Park and Ride (4 points each) are the clear hubs.

Interpretation: These are high-capacity sites designed to serve a large number of users simultaneously. Placing them in major car parks and transport interchanges is a deliberate strategy to support commuters and visitors in high-traffic areas. The key benefit of a hub is resilience; if one charger is broken or in use, others are available, leading to a better user experience.

The Spokes (Low-Density Sites):

What it shows: The majority of the sites listed have only one or two charging points. These are the "spokes" of the network.

Interpretation: The purpose of these single-charger sites is to provide maximum geographic reach. The goal is to ensure that as many towns and villages as possible have at least some access to a public charger, even if it's just one. This is crucial for encouraging EV adoption in more rural areas.

Key Insights for Your Dissertation: A Deliberate Strategy: This is not a random distribution. The chart provides strong evidence of a thought-out strategy that balances the need for high capacity in urban centers with the need for broad coverage across the region.

The Resilience vs. Coverage Trade-Off: This is the core socio-technical point. The "Hubs" provide a reliable and convenient service. The "Spokes," however, represent a potential point of failure. If the single charger in a small town like Dunkeld or Alyth is out of order, it effectively removes that entire town from the public charging map, which can be a major issue for local residents and tourists.

Policy Discussion: This plot allows you to discuss the pros and cons of this strategy. Is it better to have fewer, larger, more reliable hubs, or more, smaller, but less resilient spokes? Your dissertation can use this chart as a basis to make recommendations for the next phase of infrastructure deployment.

In [131…
```
!pip install pulp
```

```
Collecting pulp
  Downloading pulp-3.2.2-py3-none-any.whl.metadata (6.9 kB)
Downloading pulp-3.2.2-py3-none-any.whl (16.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.4/16.4 MB 69.1 MB/s eta 0:00:00
Installing collected packages: pulp
Successfully installed pulp-3.2.2
```

In [151…
```
!pip install geopandas
```

```
Collecting geopandas
  Downloading geopandas-1.1.1-py3-none-any.whl.metadata (2.3 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.11/dist-pack
ages (from geopandas) (2.0.2)
Collecting pyogrio>=0.7.2 (from geopandas)
  Downloading pyogrio-0.11.1-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (5.3
kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packag
es (from geopandas) (25.0)
Requirement already satisfied: pandas>=2.0.0 in /usr/local/lib/python3.11/dist-pa
ckages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.5.0 in /usr/local/lib/python3.11/dist-pa
ckages (from geopandas) (3.7.2)
Collecting shapely>=2.0.0 (from geopandas)
  Downloading shapely-2.1.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl.metadata (6.7 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.1
1/dist-packages (from pandas>=2.0.0->geopandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pac
kages (from pandas>=2.0.0->geopandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-p
ackages (from pandas>=2.0.0->geopandas) (2025.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages
(from pyogrio>=0.7.2->geopandas) (2025.8.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-package
s (from python-dateutil>=2.8.2->pandas>=2.0.0->geopandas) (1.17.0)
Downloading geopandas-1.1.1-py3-none-any.whl (338 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 338.4/338.4 kB 6.1 MB/s eta 0:00:00
Downloading pyogrio-0.11.1-cp311-cp311-manylinux_2_28_x86_64.whl (27.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 27.7/27.7 MB 66.5 MB/s eta 0:00:00
Downloading shapely-2.1.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.
whl (3.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 101.4 MB/s eta 0:00:00
Installing collected packages: shapely, pyogrio, geopandas
Successfully installed geopandas-1.1.1 pyogrio-0.11.1 shapely-2.1.1
```

DISCLAIMER: SOME ASPECT OF THIS CODE HAS BEEN GENERATED AND DEBIGGED USING LARGE LANGUAGE MODELS (GOOGLE GEMINI & CHATGPT)

In [ ]: