SMART CHARGING FOR SMARTER GRIDS

CIV ENG 256

Evaluating Grid Impacts of Bidirectional EV Charging in PG&E Territory

Abhinav Awasthi, Court Freund, Maya Gee, Linus Heepmann,
Celina Jakobs, Naël Oussedrat, Parker Tikson, Jinglei Zeng

Executive Summary

Electric vehicle (EV) adoption has grown rapidly throughout the US and is even more dramatic for California, where 25% of new vehicles sold are EVs and policy mandates 100% zero emission vehicle sales for new passenger cars and light duty trucks by 2035. Charging these EVs will greatly increase electricity demand, adding significant stress to the electric grid. Grid stress is a ratio between overload and capacity, where capacity is the maximum power that can reliably travel on part of the grid, and overload is any power on the grid exceeding capacity. Exceeding the capacity damages grid components and requires grid upgrades. In addition to rising charging demand, the grid, which has not seen significant load growth in decades, faces many other new challenges including growing loads from electrification and reduced demand predictability.

However, proper management of EV charging could be a significant grid asset by reducing peak loads, improving demand predictability, and possibly reducing costs. In general, managed charging refers to any framework that controls the time and rate of EV charging. Bidirectional charging, in which EVs are also able to send power back to the grid, has the potential to further improve the benefits of EVs. Senate Bill 59 in California provides state agencies with authority to require that new EVs are capable of bidirectional charging. Thus, it is important to understand how this technology could be used as a strategy to address grid strain due to EV charging.

Methodology

This project developed a methodology to analyze how installing managed bidirectional chargers in residential areas may impact grid stress, costs, and equitable access to charging, using part of Alameda County as a case study. The analysis is based on a separate research project that

estimates hourly EV charging load for a typical day in 2035, based on projected EV adoption and travel behavior for all census blocks in Alameda. A baseline scenario, in which all EV charging is unmanaged, is compared to a scenario in which managed bidirectional EV chargers can shift some of the load away from peak demand periods.

The grid stress analysis quantifies the stress this growth in EV charging would add to the grid by using capacity data provided by PG&E. This analysis focused on the grid impacts at a local distribution level and was conducted for four of the roughly 2,800 feeders (interconnected network of power lines delivering power locally) in Alameda. The maximum or peak overload on the feeder was used to approximate the required grid upgrade costs for both unmanaged and managed scenarios. The equity analysis examined how current grid strain varies across communities with different socioeconomic backgrounds and whether the benefits of a managed charging scenario were distributed evenly across communities.

Key Findings

Unmanaged EV charging in 2035 significantly increases grid stress and exceeds feeder capacity, indicating the need for major grid upgrades. Installing managed bidirectional chargers reduces peak overload on all feeders analyzed, though the level of reduction varies by feeder. Some feeders see large improvements, while others experience only limited improvements. From a cost perspective, managed bidirectional charging changes both the level and timing of distribution investments, reducing the present value of future upgrade costs for many feeders while remaining cost-increasing in feeders where charger deployment costs dominate. Feeders serving communities with higher socioeconomic burdens tend to experience higher grid stress, and managed charging reduces but does not eliminate these disparities. Feeders serving communities with higher socioeconomic burdens tend to experience higher unmanaged grid

stress, and while managed charging reduces stress across all feeders, disparities between communities remain.

Policy Implications

Overall, this Alameda case study shows that managed bidirectional EV charging can reduce grid stress and delay some grid upgrades, but it is not the sole solution. These findings help connect the quantitative results to policy by showing how Senate Bill 59 could be applied more effectively through targeted deployment of bidirectional charging.

Table of Contents

1 Introduction

The State of California has set a goal of achieving 100 percent zero emission vehicle (ZEV) sales by 2035 to advance the electrification of the transportation sector. Electric vehicles (EVs), the most common type of ZEV, currently account for about 25 percent of new car sales in California, and this share is expected to grow as the state moves towards this target.

California's electricity demand currently peaks at approximately 46 gigawatts, based on the California Energy Commission's 1-in-2 peak load forecast for 2025, and many distribution networks are already operating near capacity during high-demand periods (CAISO, 2025). The California Air Resources Board projects that statewide electricity demand could increase by 76 percent by 2045 as transportation and building sectors electrify (California Council on Science and Technology, 2025). As a result, rising EV charging demand places additional pressure on the grid, particularly when charging aligns with existing peak periods, with the most significant impacts occurring at the distribution feeder level, where existing infrastructure may require major upgrades to support increasing load.

Charging strategies can help manage the grid impacts of increasing electric vehicle (EV) adoption. Unmanaged charging occurs without grid coordination, with vehicles charging immediately at high speeds once plugged in. In contrast, managed charging controls when and how fast EVs charge. In this study, managed charging is implemented through bidirectional charging, which allows EVs to both draw from and discharge energy back to the grid, reducing peak demand and improving local grid reliability. These strategies are especially applicable in residential and workplace settings where vehicles remain parked for extended periods.

Senate Bill 59 (SB 59) directs California to coordinate a statewide strategy for expanding the zero-emission vehicle (ZEV) market, including charging infrastructure deployment. SB 59

raises the question of how bidirectional charging can help manage increasing electricity demand while improving grid reliability. In response, this study evaluates the impact of installing public bidirectional chargers on the electric grid. The analysis compares an unmanaged charging scenario with a managed public bidirectional charging scenario and quantifies differences in:

- Peak overload of a feeder, which is the maximum amount by which electricity demand exceeds the feeder's capacity during any hour.
- Infrastructure upgrade costs, required when electricity demand exceeds feeder capacity

The results of this analysis inform policy recommendations that support EV infrastructure planning aligned with SB 59. As California continues to electrify its transportation sector, understanding how bidirectional charging affects the grid at the local level is important for planning charging infrastructure.

2 Background

Electrifying transportation at scale creates new, location-specific stress on the power system. Millions of EVs will add multi-gigawatt demand, and much of that charging is likely to occur in the early evening when California's grid is already at its most vulnerable, because evening peak demand is still high while solar generation is falling (Kennedy, 2023).

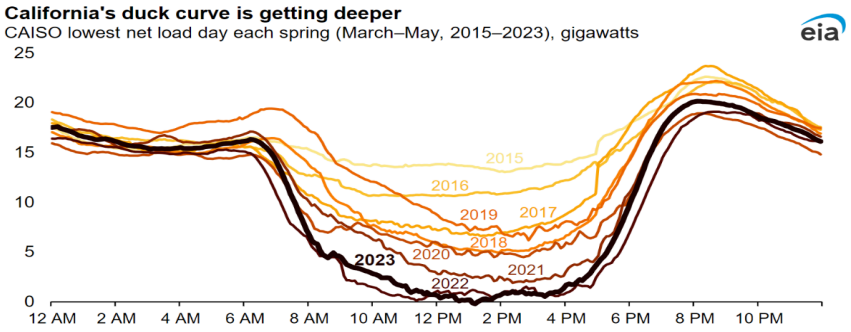


Figure 1: California duck curve 2015-2023, eia

The illustrated duck curve shows the net load, representing the total electricity demand minus the electricity generated by variable renewable energy, especially solar. It illustrates the grid stress over the course of 24 hours which results in the typical shape as there is a significant increase in non-renewable energies during nighttime.

Furthermore, residential distribution infrastructure was never built to withstand multiple Level 2 chargers on the same transformer. Studies show neighborhood feeders, whose power lines carry electricity from distribution substations to individual neighborhoods already operate at or above 100% of their rated capacity where EVs cluster (Acharige et. al. 2025). Installation of higher voltage conductors has already started to reduce the impact of higher charging rates (Jenn & Highleyman, 2022). Because level 2 chargers offer higher rate alternating current (AC) charging through 240V in residential applications, they can charge the EV faster and at a higher rate, but it usually results in significant stress on the grid (U.S. Department of Transportation, 2025). These chargers could also represent the most convenient way of being used for bidirectional charging purposes, as the 240V is the typical voltage that is needed to power the oven or dryer for example (East-West Electric Inc, 2020).

In addition to physical limitations, the administrative process of connecting high-capacity chargers can require distribution upgrades and utility reviews that can span several months, delaying public fast charging and fleet depots in precisely the areas that most need them. Therefore, EV adoption is not just a vehicle technology question; it is now a grid capacity and grid equity question. (Jenn & Highleyman, 2022).

Pacific Gas & Electric Company (PG&E) has funded multiple programs and incentives to encourage EV adoption in its service area, including rebates, charging infrastructure, preferential rates, and smart-charging technologies. For example, PG&E introduced “time-of-day” pricing to

incentivize off-peak charging and partnered with WeaveGrid to implement a smart-charging solution designed to optimize lowest-cost electricity.

Utilities and state energy regulators increasingly frame EVs as a “flexible load,” demand that can be shifted to low-cost hours with less grid strain. Shifting charging to overnight or midday periods helps smooth the overall demand curve, reduces reliance on expensive and carbon intensive natural gas peaker plants during peak hours, and makes it easier to integrate variable renewable generation such as solar and wind. Building on that idea, California is now treating EVs not only as controllable load but also as potential distributed storage: utilities are piloting vehicle-to-grid and vehicle-to-home programs with automakers and agencies such as the California Public Utilities Commission (CPUC), California Energy Commission (CEC), and California Independent System Operator (CAISO) to test whether EV batteries can act as grid assets, storing energy when it is abundant and supplying it locally when it is scarce. The broader objective is to modernize an aging distribution system so it can support large-scale electrification across households while improving reliability and managing costs (California Energy Commission, 2020). Smart charging programs can help in shifting the charging of EVs to the off-peak hours, which will reduce the load on the feeders (US Department of Energy, 2024). Thus, a balanced approach will be needed to get the most economic benefits.

Bidirectional charging is when an EV and its battery can also be an energy source for other entities. These EVs can be charged normally but also send electricity from the vehicle back to the grid or a specific load, such as a home or business. This functionality is essential for Vehicle-to-Everything (V2X) applications, where "X" can stand for different entities. In the case of this project, the battery capacity of the EV can also be used to feed the grid (V2G) or the own

home (V2H) to relieve the power grid and provide a temporary storage medium (Adegbohun et al., 2024).

California Senate Bill 59 was enacted in 2024 and primarily focused on integrating EVs into the electric grid. It allows agencies such as California Air Resources Board (CARB), the CEC, and the CPUC to require some EVs in California to have bidirectional charging abilities, provided they have a "sufficiently compelling beneficial bidirectional-capable use case". The purpose is to exploit the large storage capacities of EV batteries, making the electric grid more reliable. The bill also promotes the use of EVs as a backup power source for homes during power outages (California State Legislature, 2024).

3 Methods

To evaluate the impact of managed bidirectional electric vehicle chargers in densely populated residential areas and workplaces, we implemented a four-part methodology. First, we use a simple model and data analysis to assess the impact of managed charging on grid overload and feeder stress. Second, we conduct a cost analysis to compare the infrastructure upgrade costs required for feeders with high projected EV charging demand growth in 2030. Third, we analyze the impact of community charging solutions by measuring access and cost outcomes for lower-income communities. Fourth, we examine behavioral factors related to EV charging using existing literature, online community research, and expert interviews to understand stakeholder perspectives and identify potential challenges. And policy recommendations.

3.1 Grid Stress Analysis

EV charging is projected to cause the most stress at the distribution level (US Department of Energy, 2024); thus, we will focus our analysis of the strategy's grid impacts on the feeder level. The analysis is based on projected EV loads in 2035. This methodology is similar to Alan

Jenn's "Impact of Electric Vehicle Charging Demand on Power Distribution Grid Congestion" (Li and Jenn. To simplify our analysis, we only analyzed several PG&E feeders in Alameda County Oakland, California. The impact of EV charging on distribution feeders was modeled by combining hourly feeder base load profiles, hosting capacity, and simulated EV charging demand profiles. A baseline scenario with unmanaged EV charging was compared to one with some number of managed bidirectional chargers. The process below is repeated for each feeder being analyzed.

3.1.1 Data Integration

PG&E's Grid Resource Integration Portal (GRIP) provides hourly feeder level base load data for a high and low base load day for each month and hour. To be conservative, we used the high base load profile, denoting the hourly baseload data as $P_{base}(t)$. GRIP also provides line level hosting capacity data, which we aggregated to the feeder level. Hosting capacity is based on PG&E's ICA and estimates the maximum additional load that can be connected to a line without violating thermal limits, voltage limits, or other constraints, for a given base load condition. One day of hourly hosting capacity data is provided for each month based on two scenarios, 10th percentile and 90th percentile loading conditions (IC_{10}/IC_{90}). We aggregated this line level data to the feeder level by averaging across all lines on a feeder. This resulted in conservative feeder level hourly hosting capacity for one day of each month, $P_{IC,\,min}(t)$.

An ongoing research project by Berkeley's energy control and applications lab provided unmanaged EV charging demand at the census-block level as 15-minute power profiles for one representative day of expected EV adoption in 2025 and 2035. For the two years simulated, we approximated hourly charging power profiles by averaging the 15-minute charging profiles.

The projected hourly charging load on each feeder was calculated by summing the EV load profile from all census blocks intersecting that feeder. This analysis assumes that all EV charging demand in a given census block that intersects the feeder being analyzed, goes to that feeder, even though multiple feeders can serve a given census block. Below is a figure which shows census intersected by a given feeder (refer to Appendix A for all other feeders considered).

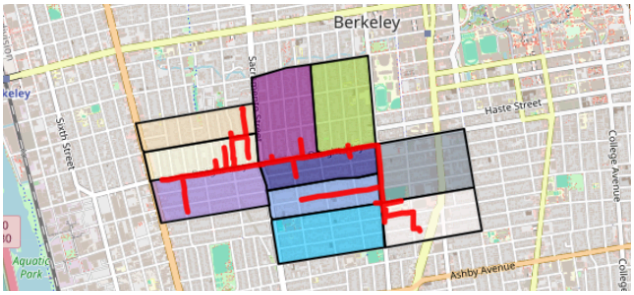


Figure 2: **Feeder 013840405 and intersecting census blocks**

The hourly EV charging demand for a given feeder based on the provided 2035 and 2025 simulation results is denoted by $P_{EV,2035}(t)$ and $P_{EV,2035}(t)$ respectively. Because the base load data from PG&E includes current EV charging, the growth in hourly EV charging power, denoted $P_{EV,growth}(t)$, was calculated by subtracting the 2025 charging power profile from the 2035, as is given by the equation below. We used this 24-hour profile as a representative day and applied it uniformly across all 12 months of grid data.

$$P_{EV\ growth}(t) = P_{EV,2035}(t) - P_{EV,2035}(t)$$

3.1.2 Baseline Scenario: Unmanaged EV Charging

In the unmanaged case, EVs charge at the charger's nominal power without active coordination. The hourly total power demand on a feeder in 2035 was then calculated by adding the growth in EV demand to the current base load, as shown by the equation below. This analysis

only considers the impact of EV load growth and does not consider any other changes in power demanded. The equations below define the hourly total load and total capacity for a given feeder.

$$P_{Total\ Load}(t) = P_{base}(t) + P_{EV\ growth}(t)$$

$$P_{Total\ Capactiy}(t) = P_{base}(t) + P_{IC,min}(t)$$

Overload quantifies how much the power on a feeder exceeds its capacity and is defined as the difference between load and capacity. Given the above definitions, the difference between total load and total capacity is reduced to the difference between EV growth and hosting capacity (the common base load terms cancel out). The equations below define our calculation of overload and peak overload.

$$P_{overload}(t) = P_{EV\ growth}(t) - P_{IC,min}(t)$$

$$Peak\ Overload = P_{overload}(t)$$

The stress metric is defined as the ratio between total load and total capacity.

$$Stress(t) = \frac{P_{Total\ Load}(t)}{P_{Total\ Capactiy}(t)}$$

$$Peak\ Stress = Stress(t)$$

This metric allows us to quantify how close a given feeder is to exceed its capacity or by what percentage it would exceed it. The analysis of By construction, Stress = 1 corresponds to the case where incremental EV demand exactly uses up the available hosting capacity. The overload is important for the cost analysis as it determines the minimum upgrades needed.

3.1.3 Managed Bidirectional Charging Scenario

To represent the impact of installing managed bidirectional chargers, a fixed fraction (f) of chargers was made to be managed bidirectional, which was treated as flexible EV load to be shifted in time, while the remaining fraction behaved as unmanaged chargers. The EV load

growth was split into unmanaged $P_{EV\ unmg}(t)$ and flexible $P_{EV\ flex}(t)$ components based on f. We then use the stress profile calculated for the baseline unmanaged scenario to construct hour-by-hour weights that give more charging to low-stress periods. This construction mimics a system of managed chargers which would reduce the charging power depending on the available power at a feeder level. The raw weight used

$$W_{raw}(t) = \frac{1}{1+(Stress_{unmg}(t))^{u}}$$

Where u > 1 is an exponent that increases the contrast between low- and high-stress hours (we empirically chose u=3). With this function, the EV demand is redistributed to off-peak hours as charging power is inversely proportional to the Stress. These weights were normalized so that the total EV charging load is unchanged. The equations below define the normalization constant C, normalized weight $W(t)$, and unmanaged $P_{EV\ unmg}(t)$ and flexible $P_{EV\ flex}(t)$ load profiles.

$$C = \frac{\sum_t W_{raw}(t)*f*P_{EV\ growth}(t)}{\sum_t f*P_{EV\ growth}(t)}$$

$$W(t) = \frac{W_{raw}(t)}{C}$$

$$P_{EV\ flex}(t) = f \bullet W(t) * P_{EV\ growth}(t)$$

$$P_{EV\ unmg}(t) = (1 - f) * P_{EV\ growth}(t)$$

$$P_{EV\ mng}(t) = P_{EV\ flex}(t) + P_{EV\ unmg}(t)$$

Using this reshaped EV profile, we again compute the stress and overloaded metrics. Importantly, the above formulation fulfills the constraint that the total energy demanded by EVs over a day is still the same as the baseline scenario

$$\int P_{EV\ mng}(t) \;=\; \int P_{EV\ growth}(t)$$

3.2 Cost Analysis

In order to evaluate the economic implications of unmanaged and managed bidirectional EV charging, we estimated the feeder level cost upgrades to meet the overloads in 2035.

Assuming PG&E will conduct distribution upgrades in discrete steps rather than one continuous increment, we adopted the stepwise upgrade model to mimic the real-life practice.

The upgrade costs were computed as a function of peak overload on the feeder, which can be defined as the maximum hourly value of EV driven overload over any given day. And so, we divided the overloads into different upgrade scenarios. Each scenario is associated with total project costs and the amount of capacity needed, representing the magnitude of the work needed (ranging from conductor replacement, transformer bank upsizing, or multi-component reinforcement, etc.).

For this study, we implemented a five-tier overload classification aligned directly with PG&E's Distribution Deferral Opportunity Report (DDOR). Following *Brockway et al.* (2022), who analyze PG&E's DDOR to estimate distribution upgrade costs by grid-need category, overloads were grouped into the following grid-need bins: <1 MW, 1–2 MW, 2–4 MW, 4–8 MW, and >8 MW. These thresholds correspond to the "grid need (MW)" categories used by PG&E to report planned circuit-level distribution investments and represent approximate decision points at which the scale and complexity of required upgrades increase.

To parameterize the piecewise cost function, we apply the **median circuit-level upgrade costs ($/kW)** reported by PG&E for each DDOR grid-need bin. Circuit-level costs are used

rather than substation-level costs to remain consistent with the feeder-level focus of this analysis. The median costs applied are:

<1 MW:	$1,875 per kW
1–2 MW:	$1,368.89 per kW
2–4 MW:	$673.35 per kW
4–8 MW:	$438.14 per kW
>8 MW:	$367.85 per kW

For each feeder, the peak overload (in kW) determines which DDOR grid-need bin applies, and the total feeder upgrade cost is calculated as the product of the peak overload and the corresponding median $/kW cost for that bin. This formulation preserves the discrete nature of utility investment decisions while allowing total upgrade costs to scale proportionally with the magnitude of the required capacity expansion **within** each tier. Mathematically, the equation will look like:

$$Overload\ Cost = \begin{cases} C_s & 0 < KW < T_s \\ C_m & T_s \leq KW < T_m \\ C_l & T_m \leq KW < T_l \\ 2C_l & KW \geq T_l \end{cases}$$

KW is overload. *Ts, Tm, Tl* are the small, medium, and large overload thresholds, respectively. *Cs, Cm, Cl* are the total project costs associated with each upgrade tier.

This DDOR-aligned tiered cost structure captures two important features of distribution planning. First, relatively small overloads can still result in high per-kilowatt upgrade costs due to fixed project components such as engineering, permitting, mobilization, and localized equipment replacement. Second, larger overloads benefit from economies of scale, as reflected in declining median $/kW costs for higher grid-need bins once major construction activities are already underway. In addition to feeder upgrades, the model also includes the capital cost of installing bidirectional chargers. Because charger installation is independent of feeder overload and occurs on a per-unit basis, we represent this cost and total cost as:

$$Bidirectional\ Charger\ Cost = Number\ of\ Chargers \cdot Cost\ of\ Each\ Charger$$

Total Cost = Upgrade Cost + Bidirectional Charger Costs

To parameterize the cost of each bidirectional charger at public near-residential stations, we start from the same family of data sources (state V2X demonstration budgets, DOE/FEMP examples, utility guidance, and commercial V2G hardware prices) but adjust them to our specific context. Public reports for early V2G pilots often quote all-in costs in the tens of thousands of dollars per site, because they combine high-power DC hardware (20–60 kW), one-off civil works, interconnection studies, and multi-year software/service contracts. For our application, by contrast, we focus on lower-power public chargers (Level 2 or small DC, 7–20 kW) deployed in clusters near residential areas, where (i) hardware prices are significantly lower, (ii) trenching, conduit, and panel upgrades are shared across multiple ports, and (iii) long-term integration and program overheads are not allocated on a per-charger basis. Synthesizing these factors, we assume a total unit cost of $5,500. This value enters linearly via the "Charger Cost" term and is consistent with a scaled-down, clustered deployment rather than isolated, pilot-style flagship installations.

Since managed bidirectional charging can also delay the timing of when those upgrades become necessary. A feeder that exceeds a threshold under unmanaged charging in 2035 may remain below that threshold under managed charging until several years later. Because utility capital expenditures are evaluated in present-value terms, deferring an upgrade produces a quantifiable financial benefit: the same project cost is incurred further in the future and is therefore discounted when measured in today's dollars. To incorporate this effect, we discount upgrade costs using PG&E's assumed cost of capital (7.5%). If an upgrade of cost C is required in year t (assumed 10 years), its present value is:

$$PV\ (C, t)\ =\ \frac{C}{(1+r)^t}$$

$$Savings\ =\ Costs\ if\ upgrades\ made\ today\ (Unmanaged) - NPV_{Managed}\ -\ NPV_{Charger}$$

Overall, this stepwise cost model aligns closely with realistic utility planning practices. It represents how infrastructure costs change as peak load varies. When integrated with the grid-stress results, its helps us to quantify how managed bidirectional charging can reduce or postpone the need for capital upgrades across the distribution system.

3.3 Equity Implications

In this section, we characterize the socioeconomic status of the communities served by the feeders analyzed in this project. This assessment builds upon the quantitative results on feeder overload and associated costs. To evaluate whether grid strain and its potential reductions are distributed equitably across communities, we use indicators from the California Office of Environmental Health Hazard Assessment (OEHHA)'s CalEnviroScreen dashboard (OEHHA, 2023). CalEnviroScreen provides percentile-based measures of socioeconomic and environmental burdens at the census-tract level, where higher percentiles correspond to communities experiencing greater disadvantages.

For our analysis, we use the population characteristics percentile indicator that combines several socioeconomic and health vulnerability indicators, including poverty, unemployment, cardiovascular disease, and educational attainment as well as other indicators. Higher population characteristics percentiles indicate communities with greater socioeconomic and health burdens and therefore greater sensitivity to utility infrastructure constraints and upgrade costs.

By computing the average population characteristics percentile for all census tracts intersecting each feeder, we evaluate whether communities with higher socioeconomic and health vulnerability experience greater grid stress and whether managed bidirectional charging

provides worse, equal, or greater benefits to these communities. To estimate feeder-level poverty burden, we averaged the CalEnviroScreen poverty percentiles of all census tracts intersecting each feeder using the following:

$$Population\ Characteristic\ Percentile_f = \frac{1}{n}\sum_{n=1}^{i} P_i$$

Where, P_i is the CalEnviroScreen Population Characteristic Percentile and n is the number of census tracts intersecting feeder

With feeder-level poverty scores established and metrics (peak overload, unmanaged upgrade costs, and managed charging upgrade costs) computed in earlier sections, we now evaluate whether feeders serving higher-poverty communities experience greater strain and whether they benefit more from managed bidirectional charging. Our analysis compares feeder-level poverty percentiles to the following previously computed metrics: Peak Overload (Unmanaged and Managed), Upgrade Costs (Unmanaged and Managed).

By examining these relationships, we assess whether the operational benefits of managed charging are aligned with the socioeconomic vulnerabilities of the communities served. This approach allows us to interpret our quantitative results through an equity lens and identify whether managed charging provides worse, equal, or better benefits to communities with higher poverty indicators.

3.4 Behavioral Interviews

To inform both the direction of this study, its implications to policy, and assess its risks to meaningfully impact future work, we interviewed four experts across academia and industry. Our selection of experts was designed to answer questions ranging from technical feasibility to industrial practicality to the importance of customer behavior and public acceptance. The

interviewees were Timothy Lipman (Ph.D., UC Berkeley), Alan Jenn (Ph.D., UC Davis), Calvin Spanbauer (Ph.D. candidate, Princeton University), and Chris Moris (Pacific Gas & Electric).

These individuals were selected to represent complementary perspectives necessary to evaluate the grid impacts of electric vehicle charging across scales and stakeholders. Timothy Lipman was selected for his extensive work on transportation electrification and vehicle-grid integration, providing an academic perspective on emerging bidirectional charging technologies and their alignment with California energy policy. Alan Jenn, Ph.D. (UC Davis) was chosen for his research on distribution-level grid congestion and EV charging demand, which closely parallels the feeder-level analytical focus of this study. Calvin Spanbauer (Princeton University, Ph.D. Candidate) was included to incorporate a research perspective focused on consumer behavior, adoption dynamics, and long-term participation barriers associated with bidirectional charging programs, which are critical to assessing the real-world feasibility of managed charging beyond technical capability. Finally, Chris Morris (Pacific Gas & Electric) was selected to represent the utility planning and operations perspective, offering insight into distribution system constraints, upgrade decision processes, and the institutional feasibility of deploying managed bidirectional charging at scale. Together, these interviews were designed to ensure that the study's modeling assumptions, behavioral considerations, and policy framing reflect both technical research and practical implementation realities.

All participants were given a summary of the project prior to the interview. Questions were prepared in advance, but interviewers chose to allow deeper dives into topics that emerged within each conversation, sometimes at the cost of less-important prepared questions. Each interview was 30 to 45 minutes in length and took place virtually on Zoom. All interviews, with the exception of Chris Moris, were recorded and transcribed for note-taking purposes. Key

findings informed both the research direction of this study as well as proposals for future policy and research.

4 Research Results & Discussion

4.1 Grid Stress Analysis

The grid stress analysis was conducted on four feeders in Alameda. Below is a plot of the hourly unmanaged and managed EV load growth profiles, as well as hosting capacity for one representative day of each month. Similar figures of all feeders analyzed are in Appendix B.

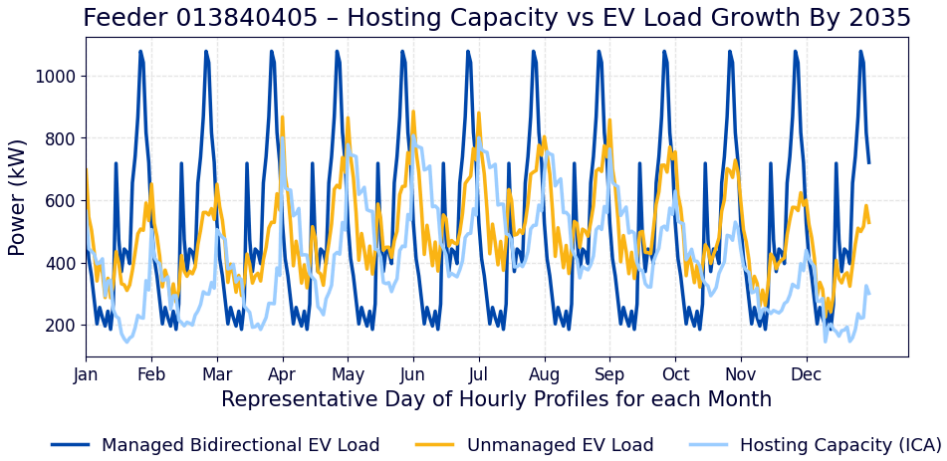


Figure 3: Hosting Capacity vs EV Load Growth By 2035

The table below summarizes the key results for all four feeders analyzed. The managed chargers were only able to completely prevent any overload for one of the four feeders.

Table 1: Grid stress analysis result summary

Feeder ID	Baseline - Unmanaged		Managed Bidirectional Chargers		Percent Reduction in Overload	Percent Reduction in Stress
	Peak Overload (kW)	Peak Stress	Peak Overload (kW)	Peak Stress		
013840405	856.65	1.92	292.00	1.30	65.91%	32.40%
012670401	961.97	1.79	667.01	1.42	30.66%	20.51%
012091102	5805.87	1.68	3055.31	1.33	47.38%	20.38%

012501108	228.39	1.03	-280.21	0.96	222.69%	7.25%

4.2 Cost Analysis

Table 2a summarizes the feeder level cost upgrades under the assumption that all investments occur in the same year, with no deferment or discounting. Under this assumption, the total cost in managed scenario can be calculated as the sum of managed cost and bidirectional cost. Which is then compared to the total unmanaged cost. Results under this assumption vary across feeders. In some cases, managed charging reduces total costs relative to the unmanaged scenario due to lower upgrade requirements. In other feeders, total costs are higher under managed charging, reflecting cases where bidirectional charger installation costs exceed the reduction in feeder upgrade costs.

Table 2a: Cost analysis results summary (No Discounting)

Feeder ID	Total Cost of Grid Upgrades		Cost of Installed Bidirectional Chargers	Total Cost Savings (Unmanaged – Managed)-Bidirectional
	Unmanaged	Managed		
01384040 5	1,606,228	547,503	676,500	382,224
01267040 1	5,381,815	3,954,483	4,829,000	-3,401,668
01209110 2	428,227	0	1,573,000	-1,144,773
01250110 8	1,803,699	1,250,651	836,000	-282,952

Now table 2b reports the result under an alternative assumption in which managed feeder and bidirectional charging can defer major upgrade requirement by 10 years. This formulation reflects how distribution of investment decisions is evaluated in practice, where utilities assess the present value of future capital requirements when making planning decisions today. Under

this assumption, managed charging is interpreted as a strategy that alters the timing and magnitude of future distribution investments rather than producing immediate cost reductions.

Table 2b: Cost analysis results summary (Managed and Bidirectional Discounted at 7.5% (via PGE) for 10 years(assumption)

Feeder ID	Total Cost of Grid Upgrades		Cost of Installed Bidirectional Chargers (NPV)	Total Cost Savings
	Unmanaged	Managed (NPV)		
013840405	1,606,228	265,712	328,001	1,012,515
012670401	5,381,815	1,918,615	2,342,364	1,120,836
012091102	428,227	0	763,367	-335,140
012501108	1,803,699	606,789	405,630	791,128

When evaluated in NPV (Net Present Value), several feeders exhibit lower total future investments under managed bidirectional charging. These outcomes arise when managed charging either avoids feeder upgrades altogether or defers them enough such that the present value of future grid upgrades and charger installation is lower than cost of upgrading unmanaged today. As table 2b suggests, the Feeders with larger unmanaged upgrade requirements tend to benefit more from this effect. This is due to the fact that substantial capital expenditure yields greater reductions in present-value cost. However, not all feeders display a positive savings value. In some cases, avoided or deferred grid upgrades are small, while the cost of installing bidirectional chargers is relatively high. This leads to higher total costs under managed charging even after discounting. This highlights that cost depends on feeder-specific factors such as upgrade size, charger deployment levels, and peak loads.

4.3 Equity Implications

Table 3 summarizes the average Population Characteristic score across the census tracts that each feeder serves, where a higher Population Characteristic score corresponds to worse socioeconomic and health burdens compared to most other communities across California.

Table 3: Feeder-Level Population Characteristic Score

Feeder ID	Feeder-level population characteristic score
013840405	42.8
012670401	70.86
012091102	64.13
012501108	12.21

Figure 4 shows feeder-level peak overload in both the unmanaged and managed charging scenarios plotted against the average CalEnviroScreen Population Characteristic percentile of the census tracts each feeder intersects.

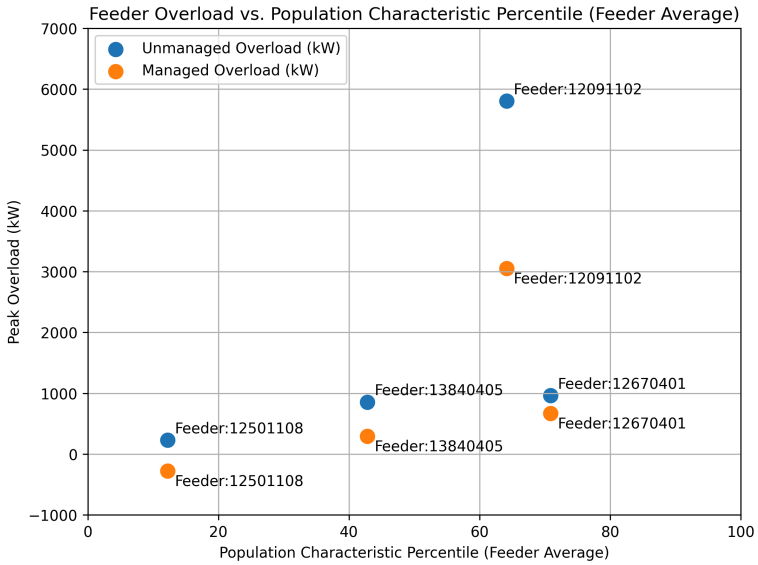


Figure 4: Feeder Overload vs. Population Characteristic Percentile

The visualization highlights differences in peak feeder overload across communities with varying population characteristic percentiles. In our sample, feeders associated with higher-percentile communities generally experience higher unmanaged peak overloads. Feeder 12091102, which has the second-highest percentile (64.13), shows the largest unmanaged overload at 5805.874 kW, while Feeder 12670401, serving the highest-percentile community

(70.86), has the second-highest unmanaged overload at 961.973 kW. In contrast, Feeders 13840405 and 12501108, which serve the lowest-percentile communities, show the lowest unmanaged peak overloads.

Regarding managed charging, all feeders experience reduced feeder strain, but the outcomes of the benefits vary. Notably, the feeder with the second-highest population characteristic percentile sees the largest decrease in peak overload, indicating that managed charging can address stress in some higher-percentile areas. However, the feeder serving the highest-percentile community shows the smallest reduction. This suggests that managed charging alone may not be sufficient enough to target the inequalities experienced by the most socioeconomically disadvantaged communities. Additionally, even with the reduced peak overload with managed charging, disparities between the communities still remain. The feeder with the lowest population characteristic percentile reaches a negative peak overload under the managed scenario, indicating that it is no longer strained, while feeders associated with higher-percentile communities are still anticipated to experience high peak stress. Regarding costs, Figure 5 below shows feeder-level upgrades costs in both the unmanaged and managed charging scenarios plotted against the average CalEnviroScreen Population Characteristic percentile.

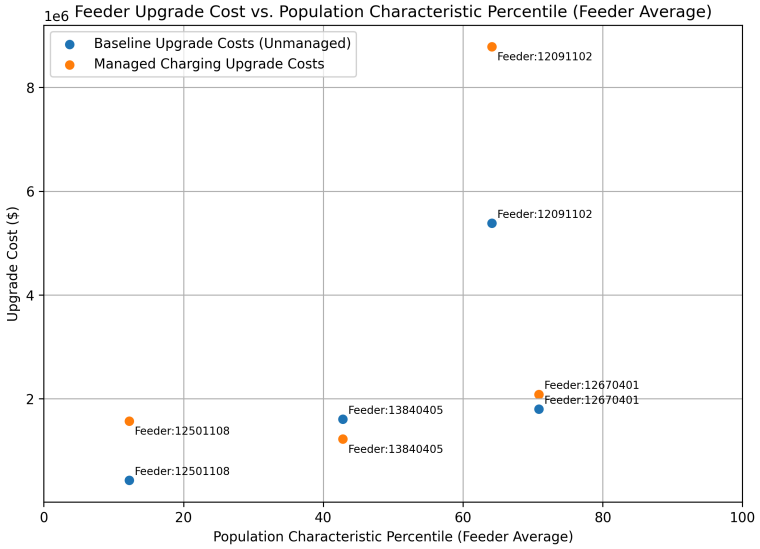


Figure 5: Upgrade Costs vs. Population Characteristic Percentile

Feeders that serve communities with higher socioeconomic burdens have higher unmanaged upgrade costs. Regarding the potential cost savings under a managed scenario, the distribution of cost benefits varies across communities with no specific trend. Some feeders experience reductions in upgrade costs, while others see increases, and there is no consistent relationship tied directly to the population characteristic percentile. Additionally, under the managed scenario, communities with greater socioeconomic burdens continue to face higher upgrade costs compared to other communities. Overall, while managed charging may provide benefits in terms of both peak overload reductions and cost savings, it does not eliminate the underlying disparities in infrastructure investment needs, suggesting that managed charging alone does not resolve cost burdens across communities.

Furthermore, these results highlight the importance of considering not only the system-level benefits of investing in bidirectional charging, but also how these investments affect communities with different socioeconomic backgrounds. Prioritizing the management of feeders that serve communities with higher population characteristic percentiles may help ensure that managed charging reduces, rather than amplifies, disparities in grid performance. However, as

shown above, implementing managed charging may not fully eliminate these differences, suggesting that additional infrastructure may be needed to support feeders that serve the most disadvantaged areas.

4.4 Expert Interviews

This section synthesizes findings from expert interviews conducted across our four expert stakeholders. The analysis identifies recurring themes that inform the interpretation of the quantitative grid-stress and cost results.

Customer behavior and adoption is core to any bidirectional solution at scale | Interviewees consistently emphasized that the effectiveness of managed or bidirectional charging depends disproportionately on customer participation, adoption, and charging behavior. Decisions around charging are often driven by affordability, convenience, perceived control, and simplicity, while complex or opaque programs reduce participation and create distrust. Customers fear losing control and agency with their vehicles and do not want to suffer the inconvenience of needing their vehicle and finding that it is not charged. Furthermore, customers often distrust utilities. Most experts agreed, however, that a commonly cited concern around battery degradation is becoming less important, as batteries are proving to be more resilient than anticipated.

Regulatory and institutional friction outweigh technical barriers | Across interviews, regulatory approval processes, tariff design, and institutional coordination were identified as more significant constraints than hardware or software readiness. Experts highlighted conservative regulatory timelines, uncertainty around incentive approval, and challenges aligning utilities, regulators, automakers, and third-party technology providers. Interviewees noted that tariff approval cycles can lag technology development by several years, incentives are often

structured as short-term pilots rather than durable programs, and unclear division of responsibility between utilities and third-party providers complicates deployment. As a result, even technically viable managed or bidirectional charging solutions face prolonged delays and fragmented implementation.

Technical barriers must be addressed, but are generally not the bottleneck | Across interviews, experts consistently indicated that the core technologies required for managed and bidirectional EV charging, such as communication protocols, control software, and power electronics, are largely mature enough to support bidirectional charging. Battery degradation is diminishing in importance due to advances in battery chemistry, thermal management, and warranty structures. However, several interviewees noted that current bidirectional charging implementations often rely on duplicative and expensive hardware, limiting cost-effectiveness at scale. As a result, while technical capability exists, the ability to deploy these technologies efficiently and reliably is constrained less by engineering limitations and more by behavioral uncertainty, regulatory processes, and institutional coordination.

Managed charging likely offers a greater (and faster) return on investment | Interviewees broadly agreed that unidirectional managed charging captures most of the near-term grid benefits at lower cost and complexity. While bidirectional charging remains technically feasible, its additional value was viewed as limited by operational uncertainty, customer availability, and coordination challenges. As a result, managed charging was consistently framed as a more scalable and dependable near-term strategy.

"Mobile" batteries pose a challenge to grid reliability | Unlike stationary storage, EVs are mobile assets whose availability varies spatially and temporally. Interviewees highlighted that utilities must plan for worst-case conditions, including peak days when vehicles may not be

plugged in or are concentrated away from constrained feeders. Because distribution planning requires high confidence in resource availability during critical hours, this mobility limits the extent to which EV batteries can be treated as firm capacity. As a result, reliance on bidirectional charging introduces additional uncertainty and reinforces the need for conservative planning assumptions when evaluating grid support potential.

5 Limitations

All EV chargers located in census blocks whose polygons intersect with a feeder geometry are assumed to be electrically supplied by that feeder. We do not model the detailed topology of secondary circuits or exact service territories within a block. Assigning all EV load from intersecting census blocks to a single feeder may over- or under-allocate demand to that feeder, especially where multiple feeders pass near the same blocks. A more detailed network model would be needed to resolve service territories.

Incremental EV load is derived from a single representative 24-hour charging profile per scenario, which is replicated across all 12 months. Seasonal and weekday/weekend variations in EV charging are not explicitly modeled. Aggregating 15-minute EV profiles to hourly averages and repeating a single day across months smooths out short-term peaks and seasonal variability. Actual peak loading events may be sharper or seasonally shifted.

We treat the difference between 2035 and 2025 EV load as the incremental demand that must be hosted by the existing feeder. The 2025 profile itself is implicitly assumed to be accommodated by the current system and ICA results. Conservative hosting capacity aggregation. ICA results are averaged across all lines and then collapsed to a single minimum value per hour, ICmin(t). This assumes that the weakest line segment limits the entire feeder and ignores potential localized headroom. Using a single feeder-average ICmin(t) ignores spatial

differences in hosting capacity along the feeder and assumes that all incremental EV load is located at the most constrained point. All stress calculations use the high base load profile rather than typical or low load conditions, to recreate a worst-case operational state.

We assumed chargers are installed on residential areas where people are willing to leave their vehicle in charge all day. Therefore, flexible load can be reshaped across hours with no constraints on individual departure times or state of charge, as long as daily energy is conserved. The managed charging scheme assumes that the chosen participation rate and control strategy are fully achievable in practice. Customer heterogeneity, non-participation, communication failures, and program design constraints are not modeled in this part of the research.

6 Policy Connections & Conclusions

While the technology of bidirectional charging may not greatly reduce the overall stress on the electricity grid because of the limited power of individual vehicles and the difficulty of coordinating large numbers of them, its real value could lay in providing specific, fast-response services at the local level. Potential was uncovered, but it became clear that the current amount of implementations would not be enough. In general, V2G is best viewed not as a replacement for large-scale power generation, but as a crucial tool for managing the complexity and stability of the local, decentralized power grid and its feeders in these areas. Because of this bottom-line, bidirectional charging should be supported in these areas to provide relief.

For further research, it should be specifically analyzed if the results conversely mean that Vehicle-to-home (V2H) is the significantly better use case for bidirectional charging. To integrate an efficient and sustainable impact on relieving the grid stress, all areas but especially those with disadvantaged groups should be focused on and be decided over what kind of change could really help there, if bidirectional charging concepts are not a cure. To integrate a policy that can

have a fundamental impact, this policy needs to target the real issues precisely by differentiating between different areas that have different needs and by being specified in its application and implementation.

The importance of customer behavior in this whole context became clear during the interviews, where the interviewees indicated that the primary challenges facing managed and bidirectional EV charging are not technical, but institutional and behavioral. Experts consistently emphasized that success depends heavily on customer participation, regulatory approval processes, tariff design, and coordination across a complex set of stakeholders. Within this context, managed unidirectional charging was generally viewed as a more reliable near-term approach, given its lower operational complexity and reduced dependence on precise vehicle availability.

Furthermore, for this implementation, the disadvantage of SB 59 was that it is too general and, to have a real impact, needs to be tailored more individually considering the above-mentioned key themes. An appropriate way could be to specify a bill like SB 59 specifically onto local and mid-stressed feeders to apply bidirectional charging to a use case, where it can have a valuable impact. To support high-stressed feeders, especially in disadvantaged communities, a different policy should be applied. This policy should focus on supporting the improvement of general grid infrastructure primarily in these areas that already need it the most. This would allow improvements to be distributed more sensibly in terms of where and when they are made, thereby better balancing the overall grid stress. In that way the general stress could be reduced, and high-stressed areas get the support to become more manageable. If these changes are applied, areas that were under high stress in the past can start to develop towards areas that can use bidirectional charging impactfully in the future.

Bibliography

Acharige, S. S., Haque, M. E., Arif, M. T., Hosseinzadeh, N., Hasan, K. N., Hossain, M., & Muttaqi, K. M. (2025). Grid integration of electric vehicles – Impact assessment and remedial measures. *Journal of Power Sources, 650*, 236697. https://doi.org/10.1016/j.jpowsour.2025.236697

Adegbohun, F., von Jouanne, A., Agamloh, E., & Yokochi, A. (2024). *A review of bidirectional charging grid support applications and battery degradation considerations*. **Energies, 17**(6), 1320. https://doi.org/10.3390/en17061320

Brockway, A., Callaway, D., & Elmallah, S. (2022). *Can distribution grid infrastructure accommodate residential electrification and electric vehicle adoption in Northern California?* (Haas School of Business Working Paper No. 327). University of California, Berkeley. https://haas.berkeley.edu/wp-content/uploads/WP327.pdf

California Council on Science and Technology. (2025, April). *Key challenges for California's energy future*. California Council on Science & Technology (CCST). https://ccst.us/reports/key-challenges-for-californias-energy-future-2/

California Public Utilities Commission, California Energy Commission, & California Independent System Operator. (2020). *Final report of the California Joint Agencies Vehicle-Grid Integration Working Group*. California Public Utilities Commission. https://www.cpuc.ca.gov/

California State Legislature. (2024, September). *Bill Text – SB-59 Menstrual Product Accessibility Act*. https://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_id=202320240SB59

California State Legislature. (2024, September 30). *SB-59 Battery electric vehicles: bidirectional capability*. https://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_id=202320240SB59

CAISO. (2025). *2025 California ISO summer loads and resources assessment report*. https://www.caiso.com/notices/2025-california-iso-summer-loads-and-resources-assessment-report-posted

East-West Electric Inc. (2020, October 17). *All you need to know about 240-volt outlets*. https://east-westelectric.com/all-you-need-to-know-about-240-volt-outlets/

Ghotge, R., Nijssen, K. P., Annema, J. A., & Lukszo, Z. (2022). *Use before you choose: What do EV drivers think about V2G after experiencing it?* **Energies, 15**(13), 4907. https://doi.org/10.3390/en15134907

Jenn, A., & Highleyman, J. (2022). *Distribution grid impacts of electric vehicles: A California case study*. **iScience, 25**(1), 103686. https://doi.org/10.1016/j.isci.2021.103686

Kennedy, R. (2023, July 5). *California's electricity duck curve is deepening*. PV Magazine. https://pv-magazine-usa.com/2023/07/05/californias-electricity-duck-curve-is-deepening/

Office of Environmental Health Hazard Assessment. (2023). *CalEnviroScreen 4.0: A new version of the California Communities Environmental Health Screening Tool*. https://oehha.ca.gov/calenviroscreen/report/calenviroscreen-40

RMI. (n.d.). *GridUp investment roadmap (2024)*. https://gridup.rmi.org/assets/GridUp-investment-roadmap.pdf

U.S. Department of Energy. (2024). *Impact of electric vehicles on the grid: Report to Congress*. https://www.energy.gov/sites/default/files/2024-10/Congressional%20Report%20EV%20Grid%20Impacts.pdf

U.S. Department of Transportation. (2025, January 31). *Electric vehicle charging speeds*. https://www.transportation.gov/rural/ev/toolkit/ev-basics/charging-speeds

Wong, Y. S., Lai, L. L., Gao, S., & Chau, K. T. (2011, July 6–9). *Stationary and mobile battery energy storage systems for smart grids*. In 2011 4th International Conference on Electric Utility Deregulation and Restructuring and Power Technologies (DRPT). IEEE. https://doi.org/10.1109/DRPT.2011.5993853

Y. Li & A. Jenn, Impact of electric vehicle charging demand on power distribution grid congestion, *Proc. Natl. Acad. Sci. U.S.A.* 121 (18) e2317599121, https://doi.org/10.1073/pnas.2317599121 (2024).

Appendix A: Feeders and Census Block Groups

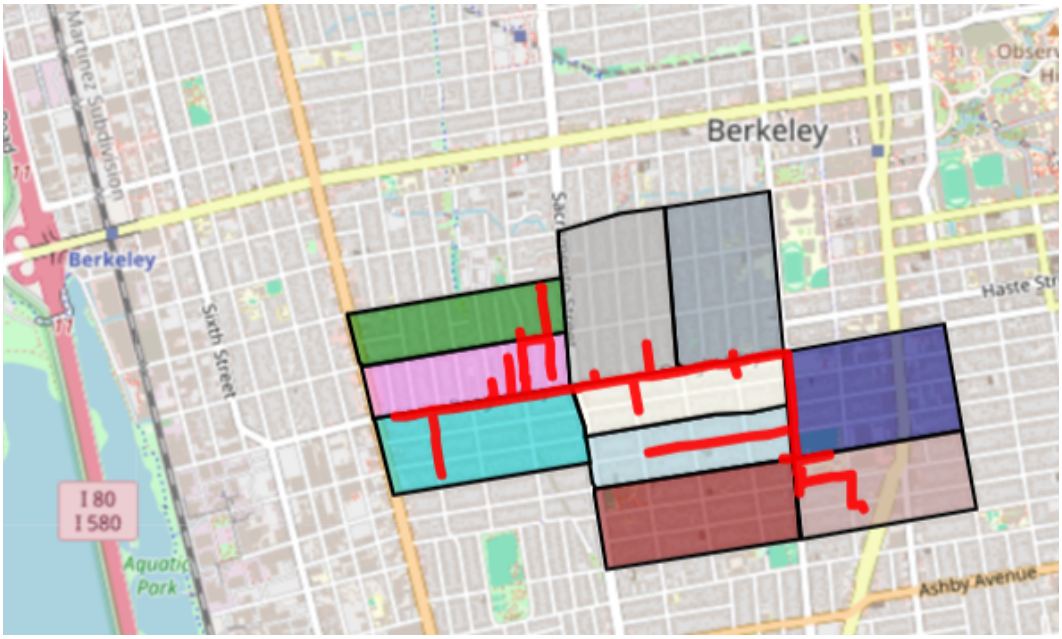


Figure A1: Feeder ID 013840405

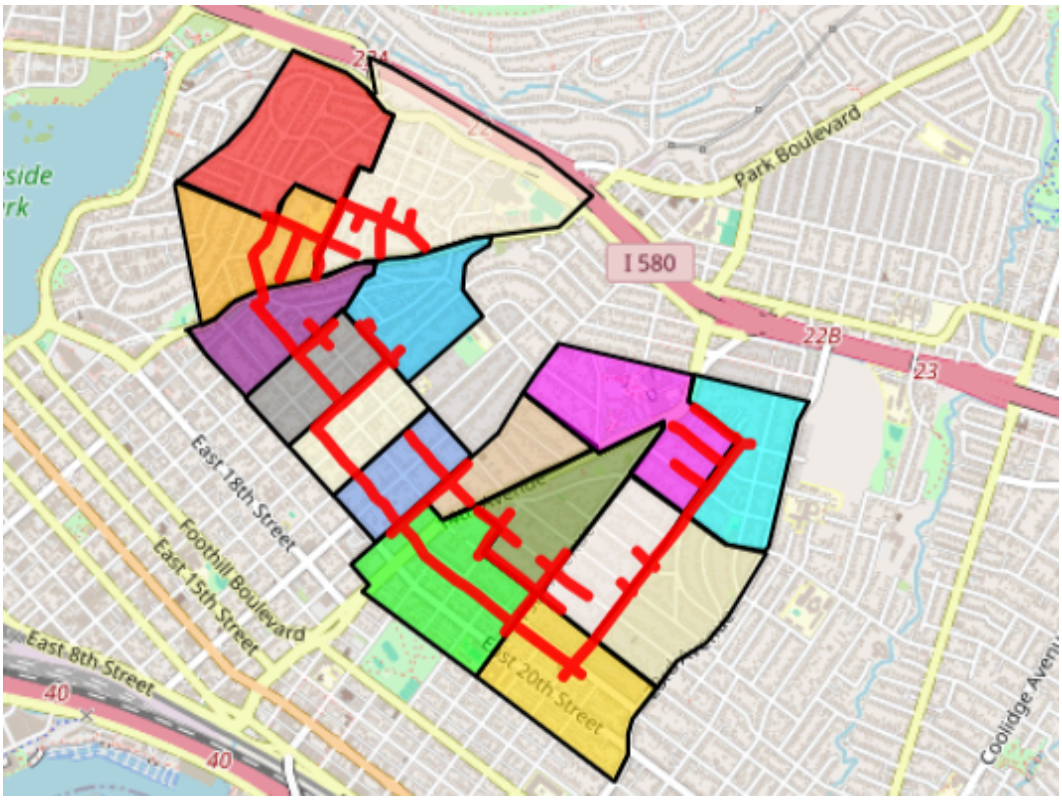


Figure A2: Feeder ID 012670401

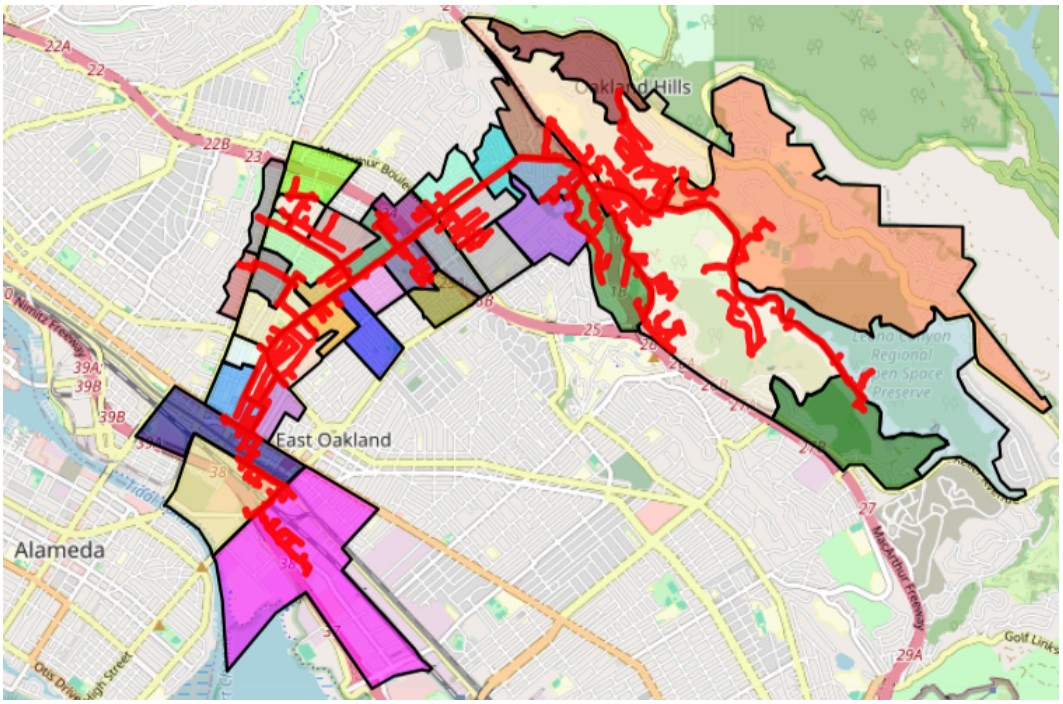


Figure A3 Feeder ID 012091102

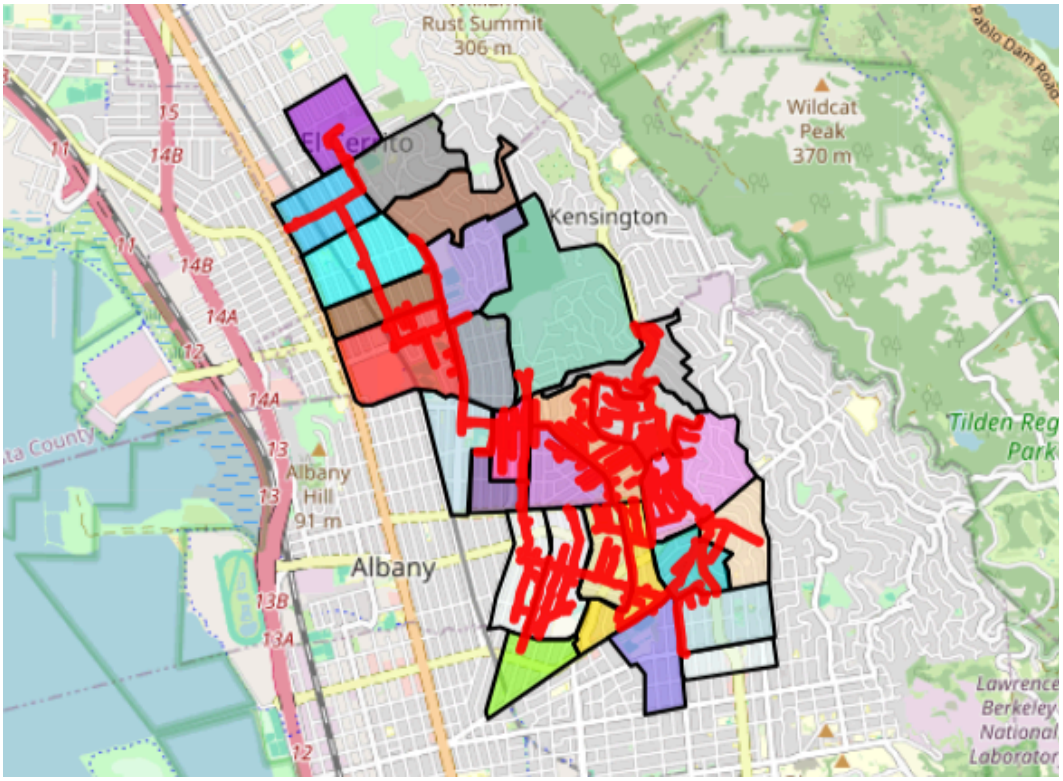
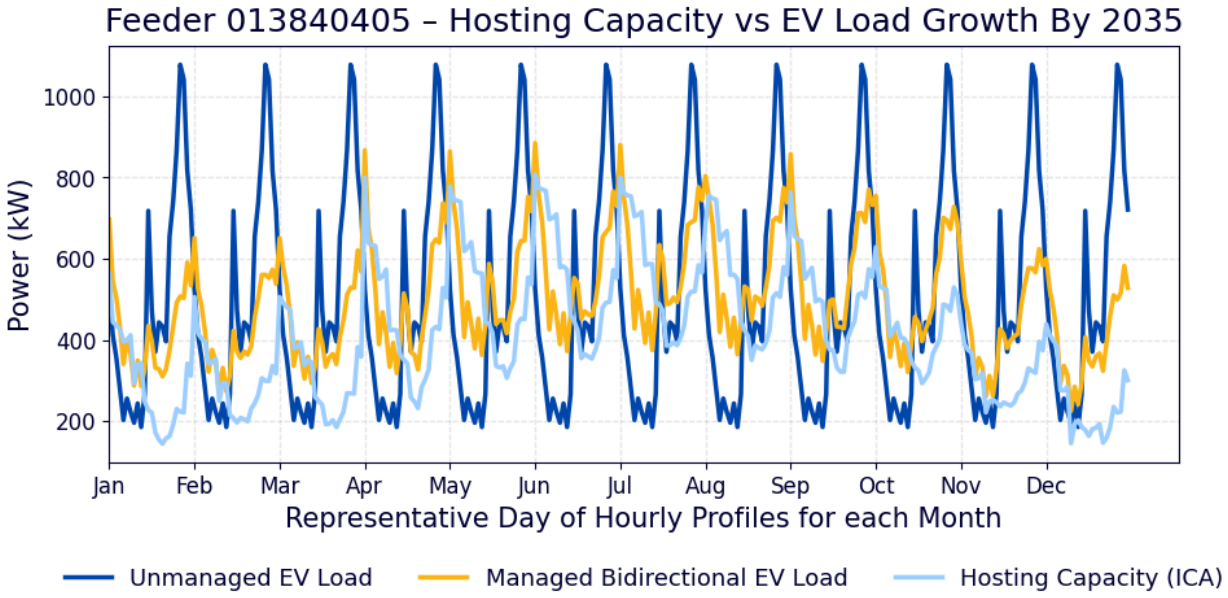
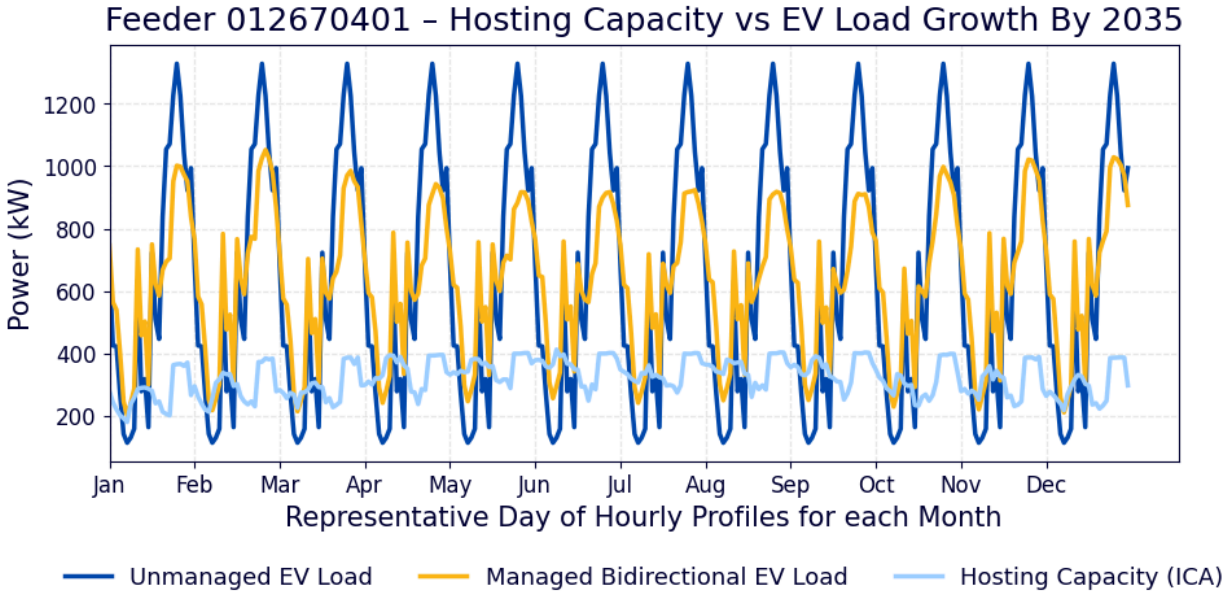
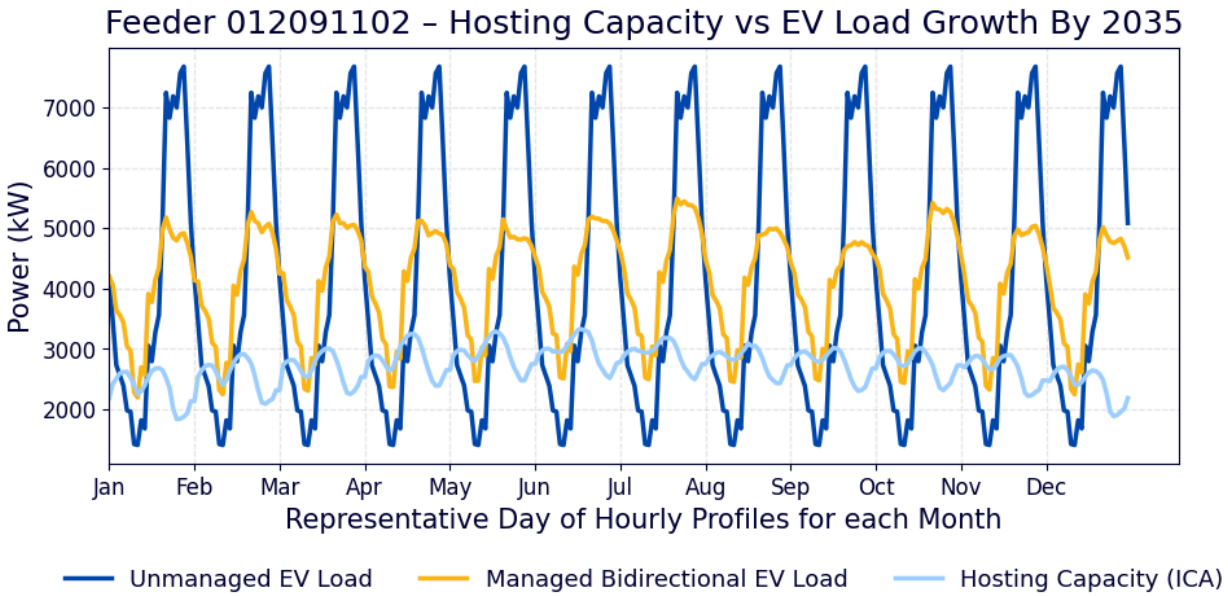


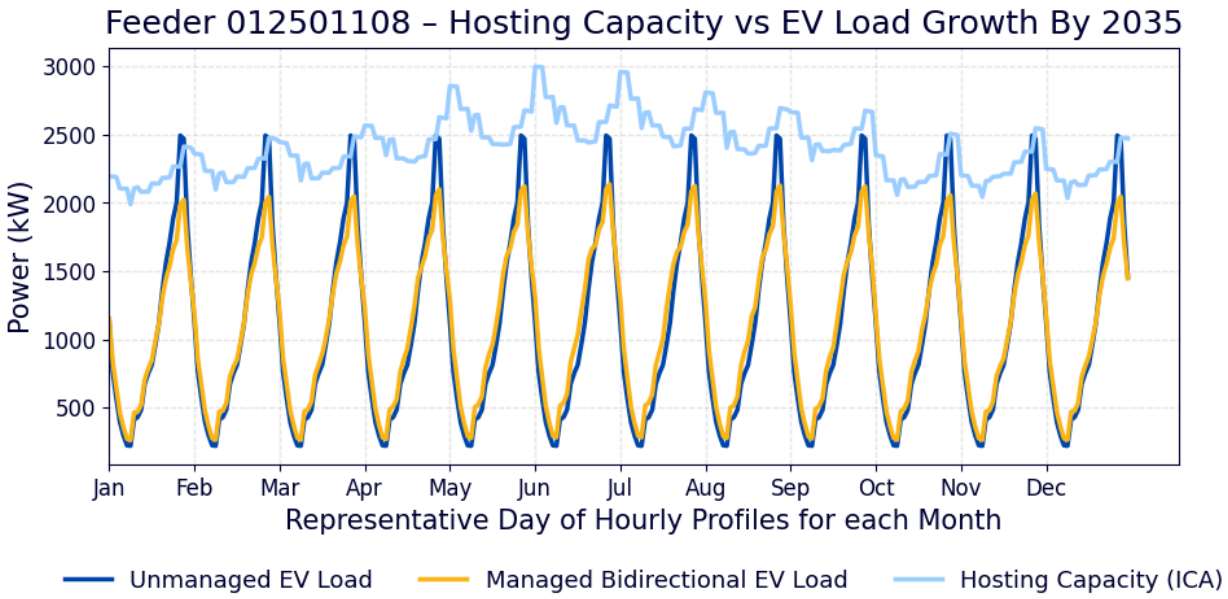
Figure A4: Feeder ID 012501108

Appendix B: Grid Stress Analysis Results






Feeder 012091102 – Hosting Capacity vs EV Load Growth By 2035
Power (kW)
7000
6000
5000
4000
3000
2000
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Representative Day of Hourly Profiles for each Month
Unmanaged EV Load
Managed Bidirectional EV Load
Hosting Capacity (ICA)


Feeder 012501108 – Hosting Capacity vs EV Load Growth By 2035
Power (kW)
3000
2500
2000
1500
1000
500
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Representative Day of Hourly Profiles for each Month
Unmanaged EV Load
Managed Bidirectional EV Load
Hosting Capacity (ICA)

Appendix C: Code - Grid Stress Analysis and Cost Analysis

```
import pandas as pd
import geopandas as gpd
import os
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import calendar
import shapely as wkt 9
                                                                                    import folium # For plots of geometry on interc
                                                                                        import matplotlib.colors a
                                                                                            impo
            13
            14 import plotly.express as px
            15 import plotly.io as pio
            16 pio.renderers.default = "colab"   # or: "notebook_connected"
            17 from plotly.graph_objs import Font
18
            19 from IPython.display import display
20
```

```
from google.colab import drive
drive.mount('/content/drive')
    3
    4 # 👇 Each user includes their own path to the shared folder starting with /content/drive/MyDrive
    5 court_base_path = "/content/drive/MyDrive/Courses/CE256 Sust Trans/[CE 256] Grid Impact Group Project/Grid

    6 nael_base_path = "/content/drive/MyDrive/[CE 256] Grid Impact Group Project/Grid Stress Analysis" 7
    8 BASE_PATH = court_base_path # Change to whoever is running the code, either: 'court_base_path' or 'nael_bas 9
   10 print("Using data folder:", BASE_PATH)
```

```
Mounted at /content/drive

Using data folder: /content/drive/MyDrive/Courses/CE256 Sust Trans/[CE 256] Grid Impact Group Project/Grid Stress Analysis
```

```
1 benchmark_feeder = '013840405' #'012670401' # '013840405' # '012091102' # Feeder in Alameda (Oakland) to P

2 # '082921104' Initial Feeder chose by Nael
```

General Functions

```
    1
    2 # ---- Shared style constants ----

    3 MAIN_COLOR = "#010133" # axis labels, ticks, border, title

    4 LINE_COLORS = ["#004AAE", "#FDB515", "#9FD1FF", "#002676",] # for curves

    5         ==
    9         Apply shared styling: colors, ticks, border, etc. """
    6         # Title and axis labels
if title is not None:
    7     def apply_unity_plot_style(ax, title=None, xlabel=None, ylabel=None):
    ax.set_title(title, color=MAIN_COLOR, fontsize=15, pad=8) if xlabel is not
0
None:
        ax.set_xlabel(xlabel, color=MAIN_COLOR, fontsize=15) if ylabel is not
1
None:
                ax.set_ylabel(ylabel, color=MAIN_COLOR, fontsize=15)

    1
            # Ticks
2           ax.tick_params(axis="x", colors=MAIN_COLOR, labelsize=12)
ax.tick_params(axis="y", colors=MAIN_COLOR, labelsize=12)
    1
3           # Border/spines

for spine in ax.spines.values():
    spine.set_color(MAIN_COLOR)
4

    1

5

    1
```

```
    # Make grid light and clean
    ax.grid(linestyle="--", alpha=0.3)

def plot_add_grid_legend_tight():
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
```

Import Data

Grid Data

```
# Import Load data for all feeders


def get_loadDf_for_feeder(feeder_id):
    feeder_loadDf = load_df_all_feeders[load_df_all_feeders['feeder_id'] == int(feeder_id)].rename(cols={'low_load_kw':'low_base_load_kw',
                                                                                                         'high_load_kw': 'high_base_load_kw'}).feeder_loadDf = feeder_loadDf.rename(columns=renameColsDict)

    return feeder_loadDf


def import_loadDf_all_feeders(base_path=BASE_PATH):

    relative_folder_path = "./Data/GRIP_ConsolidatedData_Court"
    filename = "feeder_load_profile_clean.csv"

    file_path = os.path.join(base_path, relative_folder_path, filename)
    load_df_all_feeders = pd.read_csv(file_path)
    return load_df_all_feeders
load_df_all_feeders = import_loadDf_all_feeders()
```

```
# Import metadata geodataframe for all feeders
# 1 row for each feeder 'division' 'substation' 'nom_volt_kV' 'Existing_DG' 'Queued_DG' 'shape_length' 'feeder_id' 'ge


def import_metaData_gdf_all_feeders(base_path=BASE_PATH):

    relative_folder_path = "./Data/GRIP_ConsolidatedData_Court"
    filename = "feeder_meta_clean_gdf.gpkg"
    file_path = os.path.join(base_path, relative_folder_path, filename)
    gdf_all_feeders = gpd.read_file(file_path)
    return gdf_all_feeders
```

```
feederMetadataGdf = import_metaData_gdf_all_feeders()
display(feederMetadataGdf.head())
```

Show hidden output

```
# FEEDER/LINES ICA Data Import and Average functions

def get_feederAvgIcaDf(feeder_id):
    linesIcaDf = import_feederLineIcaDf_from_parquet(feeder_id)
    # Check that there is only one feeder_id in df
```

```
    feederIcaDf = avg_linesIca_for_feederIca(linesIcaDf)

    return feederIcaDf

def avg_linesIca_for_feederIca(linesIcaDf):
    ic_cols = ['IC10_Thermal_KW', 'IC10_Voltage_KW',
                       'IC90_Thermal_KW', 'IC90_Voltage_KW']

    # --- Step 2: group by feeder, month, and hour and take the mean ---
    feederIcaDf = (
            linesIcaDf
            .groupby(['feeder_id', 'division', 'month', 'hour'])[ic_cols]
            .mean()
            .reset_index()
    )

    # Rename ic_cols so they are called avg
    ic_cols_new =           ['avg_IC10_Thermal_KW', 'avg_IC10_Voltage_KW',
                                    'avg_IC90_Thermal_KW', 'avg_IC90_Voltage_KW']

    feederIcaDf = feederIcaDf.rename(columns = dict(zip(ic_cols, ic_cols_new)))

    # create new columns with min of the ic_cols_new
          feederIcaDf['minAvg_IC_KW'] = feederIcaDf[ic_cols_new].min(axis=1) 34
    # check result
    return feederIcaDf




# Import ICA data from parquet files
def import_feederLineIcaDf_from_parquet(feeder_id, base_path=BASE_PATH):
    """
    get_feeder_ICA_df_from_parquet

    :feeder_id: Feeder ID whose ICA data you want
    :folder_path: Path to the folder in Google Drive that holds the .parquet files
    :return: Pandas DataFrame
    """
    relative_folder_path = "./Data/GRIP_ConsolidatedData_Court/ICA_Load_CLEAN_PARQUET_v4"
    filename = f"{feeder_id}.parquet"
    file_path = os.path.join(base_path, relative_folder_path, filename)
    if not os.path.exists(file_path):
          raise FileNotFoundError(f"File not found: {file_path}") 55
    df = pd.read_parquet(file_path)
    # rint(f"Loaded {filename} with {len(df)} rows and {len(df.columns)} columns.")
    return df
```

```
# TEST & DISPLYA ICA DATA
feederLinesIcaDf = import_feederLineIcaDf_from_parquet('012011108')
print("feederLinesIcaDf")
display(feederLinesIcaDf.head())
      5

feederAvgIcaDf = get_feederAvgIcaDf(benchmark_feeder)
print("\n\nfeederAvgIcaDf")
display(feederAvgIcaDf.head(25))
```

Show hidden output

Census from Thibaud

```
# Import Census tract data
def import_census_block_gdf_for_charging():
```

```
    relative_folder_path = "./Data/Geography Files"
    filename = "location_str_to_geoid_mapping.shp"
    file_path = os.path.join(BASE_PATH, relative_folder_path, filename)
    gdf = gpd.read_file(file_path)
    return gdf
```

```
evCensusGdf = import_census_block_gdf_for_charging()
display(evCensusGdf.head())
```

	GEOID	GEOID_STR	INTPTLAT	INTPTLON	geometry
0	060014232002	2 (Tract 4232, Alameda, CA)	37.862497	-122.293241	POLYGON ((-122.29678 37.86529, -122.29581 37.8...
1	060590423361	1 (Tract 423.36, Orange, CA)	33.545621	-117.699173	POLYGON ((-117.70528 33.54996, -117.70506 33.5...
2	060014043002	2 (Tract 4043, Alameda, CA)	37.844708	-122.241138	POLYGON ((-122.24813 37.84139, -122.24749 37.8...
3	060133851001	1 (Tract 3851, Contra Costa, CA)	37.924964	-122.298942	POLYGON ((-122.30682 37.93159, -122.30682 37.9...
4	060952512002	2 (Tract 2512, Solano, CA)	38.104862	-122.237457	POLYGON ((-122.24357 38.10294, -122.24353 38.1...

```
# Extract geodataframe for all block groups intersecting a given feeder
def blocks_intersecting_feeder_df(feeder_id, gdf_feeders, gdf_blocks):
    feeder_row = get_feeder_row(gdf_feeders, feeder_id)
    blocks_proj = reproject_blocks_to_feeder(gdf_blocks, gdf_feeders)
    feeder_geom = feeder_row.geometry.iloc[0]
    intersecting_blocks = get_intersecting_blocks(blocks_proj, feeder_geom)
    print(f"Found {len(intersecting_blocks)} blocks intersecting feeder {feeder_id}")
    # display(intersecting_blocks)
    save_intersecting_blocks_df(intersecting_blocks, feeder_id)
      print("saved intersecting blocks") # saved intersecting 11

    return intersecting_blocks


# 1) tiny helper: get feeder row
def get_feeder_row(gdf_feeders, feeder_id, feeder_id_col="feeder_id"):
    feeder_row = gdf_feeders[gdf_feeders[feeder_id_col] == feeder_id]
    if feeder_row.empty:
        raise ValueError(f"No feeder found with ID {feeder_id}")
    return feeder_row

# 2) tiny helper: reproject blocks to feeder CRS
def reproject_blocks_to_feeder(gdf_blocks, gdf_feeders):
    if gdf_feeders.crs is None:
        raise ValueError("gdf_feeders has no CRS set")
    if gdf_blocks.crs is None:
        raise ValueError("gdf_blocks has no CRS set")
    return gdf_blocks.to_crs(gdf_feeders.crs)

# 3) tiny helper: get intersecting blocks (same CRS assumed)
def get_intersecting_blocks(blocks_same_crs, feeder_geom):
    mask = blocks_same_crs.intersects(feeder_geom)
    return blocks_same_crs[mask].copy()

def save_intersecting_blocks_df(intersecting_blocks, feeder_id):
    # name output file based on feeder idf
    output_file = f"census_block_groups_intersecting_feeder_{feeder_id}.csv"
    save_path = os.path.join(court_base_path, "Cost_Input_from_Grid_Stress_Analysis", output_file)
    intersecting_blocks.to_csv(save_path, index=False)

intersectingBlocks = blocks_intersecting_feeder_df(benchmark_feeder, feederMetadataGdf, evCensusGdf)
display(intersectingBlocks)
```

Found 10 blocks intersecting feeder 013840405 saved intersecting blocks

	GEOID	GEOID_STR	INTPTLAT	INTPTLON	geometry
896	060014234002	2 (Tract 4234, Alameda, CA)	37.860138	-122.276163	POLYGON ((563278.556 4190579.647, 563292.877 4...
3801	060014234003	3 (Tract 4234, Alameda, CA)	37.857906	-122.275808	POLYGON ((563311.814 4190384.95, 563318.232 41...
5649	060014234001	1 (Tract 4234, Alameda, CA)	37.861911	-122.276458	POLYGON ((563224.748 4190769.969, 563239.861 4...
7340	060014233001	1 (Tract 4233, Alameda, CA)	37.860361	-122.285013	POLYGON ((562484.589 4190642.139, 562491.084 4...
7439	060014235001	1 (Tract 4235, Alameda, CA)	37.861996	-122.268418	POLYGON ((564021.141 4190903.41, 564028.349 41...
8934	060014231004	4 (Tract 4231, Alameda, CA)	37.862553	-122.285632	POLYGON ((562443.68 4190839.222, 562452.992 41...
18190	060014230002	2 (Tract 4230, Alameda, CA)	37.865788	-122.274707	POLYGON ((563569.233 4191440.155, 563589.35 41...
23073	060014235002	2 (Tract 4235, Alameda, CA)	37.858856	-122.267924	POLYGON ((564059.339 4190506.144, 564066.301 4...
24458	060014231003	3 (Tract 4231, Alameda, CA)	37.864342	-122.286005	POLYGON ((562387.662 4191033.638, 562393.368 4...
25031	060014230003	3 (Tract 4230, Alameda, CA)	37.865174	-122.279216	POLYGON ((563171.333 4191343.989, 563186.334 4...

Plots

```
def plot_feeder_and_blocks(feeder_id, gdf_feeders, gdf_blocks):
    feeder_row = get_feeder_row(gdf_feeders, feeder_id)
    blocks_proj = reproject_blocks_to_feeder(gdf_blocks, gdf_feeders)
feeder_geom = feeder_row.geometry.iloc[0]
    blocks_hit = get_intersecting_blocks(blocks_proj, feeder_geom)

    fig, ax = plt.subplots(figsize=(8, 8))

    if not blocks_hit.empty:
        blocks_hit.plot(ax=ax, facecolor="lightgray", edgecolor="gray", linewidth=0.5) feeder_row.plot(ax=ax,
color="red", linewidth=2)
        ax.set_title(f"Feeder {feeder_id} and intersecting census blocks") ax.set_axis_off()
        plt.tight_layout()
plt.show()

plot_feeder_and_blocks(benchmark_feeder, feederMetadataGdf, intersectingBlocks)
```

Show hidden output

```
# Create plot of overlapping blocks and feeder on interactive map
def folium_plot_feeder_and_blocks(feeder_id, gdf_feeders, gdf_blocks, unique_colors=True
    # put both in WGS84 for web map
    feeders_wgs = gdf_feeders.to_crs(epsg=4326) blocks_wgs
= gdf_blocks.to_crs(epsg=4326)

    feeder_row = get_feeder_row(feeders_wgs, feeder_id)
feeder_geom = feeder_row.geometry.iloc[0]
    blocks_hit = get_intersecting_blocks(blocks_wgs, feeder_geom) center =
feeder_geom.representative_point().coords[0][::-1]
    m = folium.Map(location=center, zoom_start=12, tiles="OpenStreetMap",
                   width = "800px",
                height="500px")

    # style
    if unique_colors and not blocks_hit.empty:
        colors = random.sample(list(mcolors.CSS4_COLORS.keys()), len(blocks_hit))
blocks_hit = blocks_hit.assign(_color=colors)
        style_fn = lambda feat: {
            "fillColor": feat["properties"]["_color"],
```

```
                "color": "black",
                "weight": 2,
                "fillOpacity": 0.55,
            }
        else:
            style_fn = lambda feat: { "fillColor":
                "#999999", "color": "black",
                "weight": 2,
                "fillOpacity": 0.45,
            }

        folium.GeoJson( blocks_hit,
            name="Intersecting blocks",
            style_function=style_fn,
        ).add_to(m)

    folium.GeoJson( feeder_row,
        name="Feeder",
        style_function=lambda x: {"color": "red", "weight": 5, "opacity": 0.9},
    ).add_to(m)

    folium.LayerControl().add_to(m)
    print(f"Feeder ID {feeder_id}")
    display(m)
    return m
```

EV Projection Data

```
import ast
import re
def import_ev_projection_df(year):
    if year == "full adoption":
        relative_folder_path = "./Data/sim_20251117_alameda_full_adoption" elif year == "2035":
        relative_folder_path = "./Data/sim_20251117_alameda_2035_adoption" elif year == "2025":
            relative_folder_path = "./Data/sim_20251117_alameda_2025_adoption"

    print(year)

    filename = "ev_load_curves.csv"

    file_path = os.path.join(BASE_PATH, relative_folder_path, filename) df = pd.read_csv(file_path)
    # Convert string "[...]" → Python list of floats if "load_curve" in df.columns:
        df["load_curve"] = df["load_curve"].apply(lambda x: ast.literal_eval(x) if isinstance(x, str) else return df

ev_projection_df = import_ev_projection_df("2035")
display(ev_projection_df.head())
```

2
035

	geoid	charger_type	number_of_sessions	load_curve
0	0 (Tract 9900, Alameda, CA)	Public_HD_Long_Duration_DCFC_150_kW	4	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
1	0 (Tract 9900, Alameda, CA)	Public_HD_Long_Duration_DCFC_350_kW	4	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
2	0 (Tract 9900, Alameda, CA)	Public_MD_Long_Duration_DCFC_150_kW	2	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
3	0 (Tract 9900, Alameda, CA)	Public_MD_Long_Duration_DCFC_50_kW	3	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...
4	0 (Tract 9900, Alameda, CA)	Public_MD_Long_Duration_L2_11_kW	9	[11.2, 11.2, 11.2, 11.2, 11.2, 11.2, 11.2, 11....

Aggregate Load Data for Blocks

```
1 # Extract block number (including decimals) ---
2 def extract_block(geoid_str):
3       if not isinstance(geoid_str, str):
4             return None
5       match = re.search(r"block\s+([\d.]+)", geoid_str)
6       if match:
7             try:
8                   return float(match.group(1))
9             except ValueError:
10                  return None
11      return None
12
13
14 # extract all rows of chargning data for a given census block
15 def filter_by_block(df, block_number):
16     df =
df.copy() 17
18      # some geoids are just called out of region, remove those
19        df = df[df["geoid"] != "Out of Region"] 20
21      # df["block_num_int"] = df["geoid"].str.extract(r"block\s+(\d+)").astype(int)
22        return df[df["block_float"] == float(block_number)] 23
24
25 def get_sum_charge_curve_lists(df_subset):
26      """
27      Sum multiple load_curve lists into one aggregate array.
28      """
29      if "load_curve" not in df_subset.columns:
30        raise ValueError("DataFrame must contain a 'load_curve' column.") 31
32      # Convert all to numpy arrays in case some are still lists
33        curves = [np.array(curve, dtype=float) for curve in df_subset["load_curve"]] 34
35      if len(curves) == 0:
36            return []
37
38      total_curve = np.sum(curves, axis=0)
39      return total_curve.tolist()
40
41
42 def get_hourly_load_curve_from_15min(load_15min, method):
43      """
44      Convert a 15-minute load curve (length 96) to an hourly load curve (length 24).
45        Either by average the 4 quarter-hours or take the max. 46
47      Inputs
48            loaad_15min : list or array-like 15-minute load values, expected length = 96.
49        method : str, "avg" or "max" 50
51      Returns: list of Hourly load curve (length 24).
52      """
53        arr = np.array(load_15min, dtype=float) 54
55      if len(arr) % 4 != 0:
```

```
                raise ValueError(f"Expected length multiple of 4 (e.g. 96), got {len(arr)}")

            # reshape to (24, 4): 24 hours, 4 quarters each hourly_blocks = arr.reshape(-1, 4)

            if method == "avg":
                hourly = hourly_blocks.mean(axis=1) elif method == "max":
                hourly = hourly_blocks.max(axis=1) else:
                raise ValueError("method must be 'avg' or 'max'")

            return hourly.tolist()

# TESTING
# ev_projection_df = import_ev_projection_df()
# display(ev_projection_df)
# block_df = filter_by_block(ev_projection_df, '4059.02')
# display(block_df)
# type(block_df["load_curve"].iloc[1])
# sum = get_sum_load_curve_lists(block_df)
# print(sum)
# hrlyLoadCurve = get_hourly_load_curve_from_15min(sum, "avg")
# print(hrlyLoadCurve)
```

```
# Function to get the total EV projected charging demand from all blocks intersecting a Feeder
def get_aggregate_evLoadDf_for_blocksDf(blocksGdf, evLoadDf):
    """
    blockssGdf is all the instersecting census blocks for a feeder
    """
    evLoadDf_matchingBlocks = pd.DataFrame()
    for index, row in blocksGdf.iterrows():
        block_geoID = row["GEOID_STR"]
        # Extract all rows of charging data that match the geoID of this block
      evLoadDf_forBlock = evLoadDf[evLoadDf["geoid"] == block_geoID]
        # display(evLoadDf_forBlock)
        # append block_df to projectionDf
      evLoadDf_matchingBlocks = pd.concat([evLoadDf_matchingBlocks, evLoadDf_forBlock], ignore_index=True
    # 1) sum all 15-min load curves across those rows (length 96)
    sumChargeLoadList = get_sum_charge_curve_lists(
        evLoadDf_matchingBlocks
      ) # or your sum_load_curve_lists()
    # 2) convert to hourly avg and hourly peak (each length 24)
    avgHrlyLoadCurve = get_hourly_load_curve_from_15min(sumChargeLoadList, method="avg")
    peakHrlyLoadCurve = get_hourly_load_curve_from_15min(
        sumChargeLoadList, method="max"
    )

    # 3) build final 24-row dataframe
    out_df = pd.DataFrame(
        {
            "hour": list(range(24)),
            "hrlyAvgEVLoadKW": avgHrlyLoadCurve,
            "peakHrlyEVLoadKW": peakHrlyLoadCurve,
        }
    )

    return out_df

# to get EV load growth for a given block subtract the estimated current demand from projected future charg
def get_projected_evLoadGrowthDf_for_blocksDf(blocksGdf, evLoadNowDf, evLoadProjectedDf):
    nowDf = get_aggregate_evLoadDf_for_blocksDf(blocksGdf, evLoadNowDf)
```

```
41      projectedDf = get_aggregate_evLoadDf_for_blocksDf(blocksGdf, evLoadProjectedDf)
```

```
    # growthDf = projectedDf - nowDf
    growthDf = projectedDf.set_index("hour").subtract(nowDf.set_index("hour")) growthDf = growthDf.reset_index()
    return growthDf

# display(get_aggregate_evLoadDf_for_blocksDf(intersectingBlocks, ev_projection_df))
evLoad2035Df = import_ev_projection_df("2035")
evLoad2025Df = import_ev_projection_df("2025")
evBlocksLoadGrowthDf = get_projected_evLoadGrowthDf_for_blocksDf(intersectingBlocks, evLoad2025Df, evLoad20
display(evBlocksLoadGrowthDf)
```

Show hidden output

```
display(evLoad2035Df)
display(evLoad2025Df)
```

Show hidden output

Plot Functions

```
def plot_feeder_ica_concatenated(
                                                    43
                                                    44
                                                    45
):                                                  46
```

feeder_id,ic_cols=None,legend_labels=None,

    """

Plot ICA columns over a concatenated time line: each month contributes 24 hours on the x-axis.
Uses 'pretty' color scheme and nicer legend labels.
"""
f

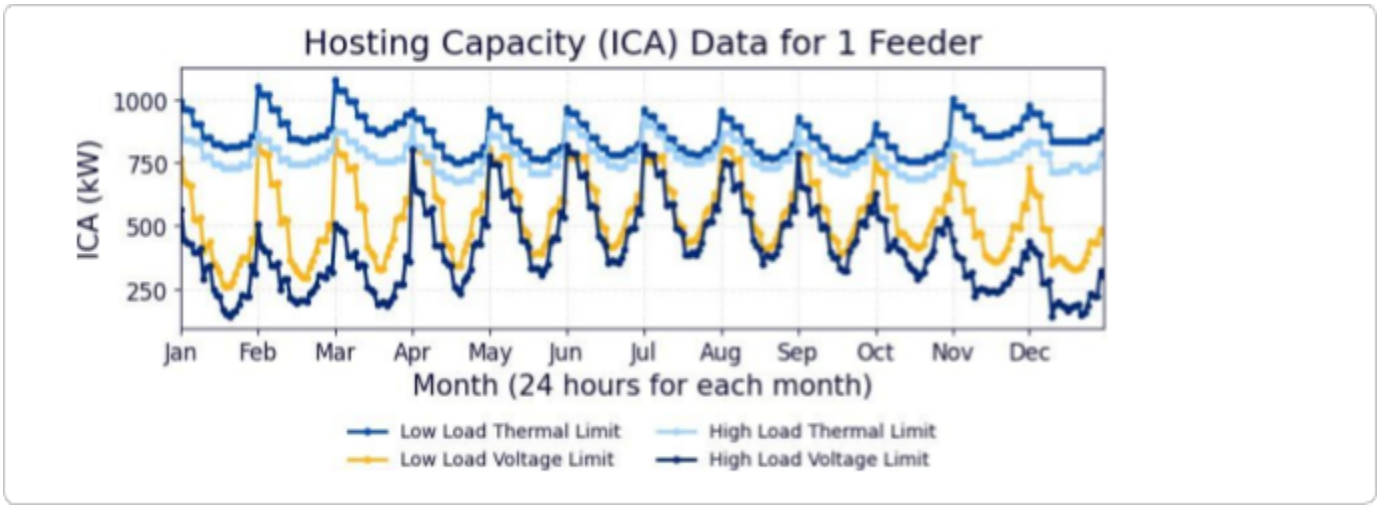
e

e

```
derIcaDf = get_feederAvgIcaDf(feeder_id) # Default ICA columns and legend labels
if ic_cols is None: ic_cols = [
    'avg_IC1
```

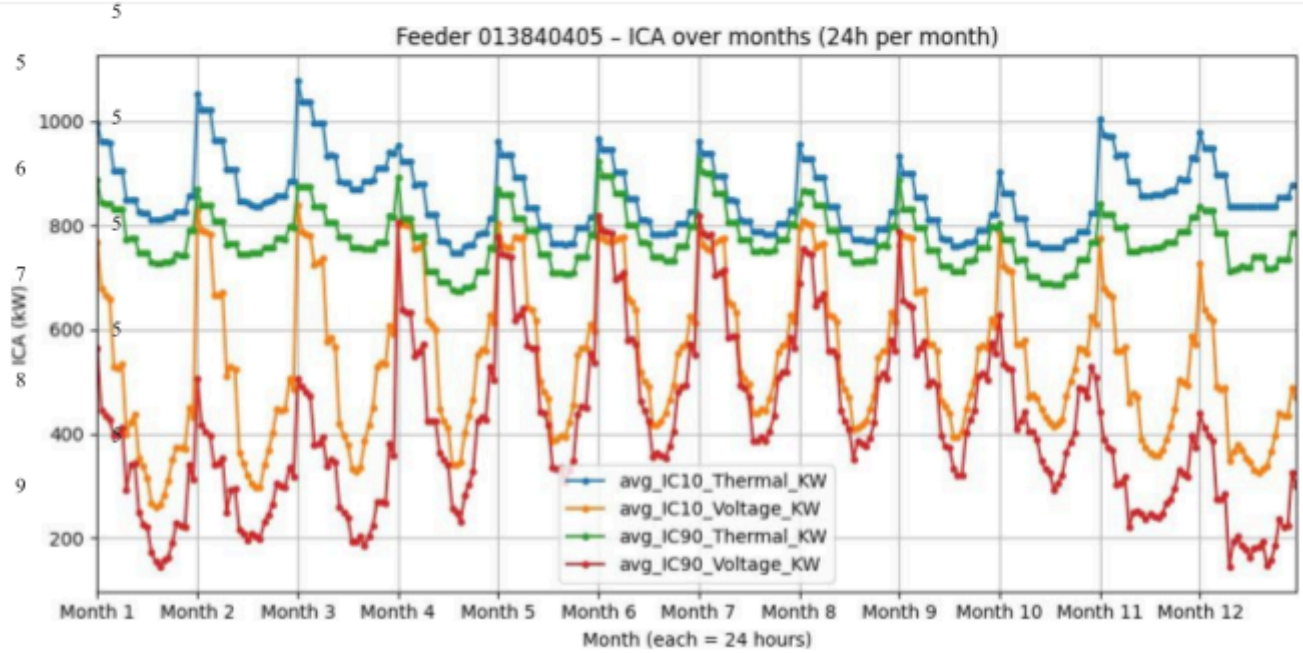
```
                    o
                    a
                    d
                    V
                    o
                    l
                    t
                    a
                    g
                    e
                    L
                    i
                    m
                    i
                    t
                    "
                    ]

# filter to this feeder
df_f =
feederIcaDf[feederIca
Df['feeder_id'] ==
feeder_id].copy() if
df_f.empty:
    p
    r
    i
    n
    t
    (
    f
    "
    N
    o
    d
    a
    t
    a
    f
    o
    r
    f
    e
    e
    d
    e
    r
    {
    f
    e
    e
    d
    e
    r
    _
    i
    d
    }
    "
    )
    r
    e
    t
    u
    r
    n

# make sure months are in order
months = sorted(df_f['month'].unique())

#
w
e
w
i
l
l
b
u
i
l
d
a
r
r
a
y
s
i
n
m
o
n
t
h
o
r
d
e
r
a
l
l
_
h
o
u
r
s
=
[
]
data_by_col = {col: [] for col in ic_cols}

for mi, m in enumerate(months):
    df_m = df_f[df_f['month'] == m].sort_values('hour')

    # x positions for this month: shift by 24 * month_index
```

```
            base = mi * 24
            x_this_month = base + df_m['hour'].to_numpy()
            all_hours.append(x_this_month)

            for col in ic_cols:
                data_by_col[col].append(df_m[col].to_numpy())

    # concatenate
    x = np.concatenate(all_hours)
    for col in ic_cols:
      data_by_col[col] = np.concatenate(data_by_col[col]) 58
    # plot
    fig, ax = plt.subplots(figsize=(8, 4))
    for i, col in enumerate(ic_cols):
            # skip if column is missing
            if col not in df_f.columns:
                continue

            label = legend_labels[i] if i < len(legend_labels) else col
            color = LINE_COLORS[i % len(LINE_COLORS)]

                                                                                                    ax.plot(
                                                                                                        x,
                                                                                                data_by_col[col],
                                                                                                    marker='.',
                                                                                                  linewidth=2.0,
                                                                                                    label=label,
                                                                                                    color=color,
            )

    # x-ticks at the start of each month block
    month_xticks = [i * 24 for i in range(len(months))]
    # month_xticklabels = [f"Month {m}" for m in months]
    month_xticklabels = [calendar.month_abbr[m] for m in months] # <-- NEW
    ax.set_xticks(month_xticks)
      ax.set_xticklabels(month_xticklabels, rotation=0) # , fontsize=12) 84
    # axis limits
    ax.set_xlim(0, x[-1])

                                                                                                        # apply shared styling
                                                                                    title = f"Hosting Capacity (ICA) Data for 1 Feeder"
                                                                                                    apply_pretty_plot_style(
                                                                                                        ax,
                                                                                                    title=title,
                                                                                xlabel="Month (24 hours for each month)",
                                                                                                    ylabel="ICA (kW)",
    )

    # legend styling (match MAIN_COLOR)
    legend = ax.legend(
            loc="upper center",
            bbox_to_anchor=(0.5, -0.3),                # x=0.5 centers it, y is distance below plot
            ncol=2,                                      # good if you have multiple legend entries
            frameon=False)
    # legend = ax.legend(ncol=1, fontsize=12, frameon=True) # , bbox_to_anchor=(0.5, -0.15))
    # legend.set_bbox_to_anchor((0.44, 0.6), transform=ax.transAxes)

    for text in legend.get_texts():
            text.set_color(MAIN_COLOR)

    plt.tight_layout()
    plt.show()
plot_feeder_ica_concatenated(benchmark_feeder)
```




```
# PLOT AVG ICA DATA FOR A FEEDER


def plot_feeder_ica_concatenated(feeder_id):
    """
    Plot 4 ICA columns over a concatenated timeline:
    each month contributes 24 hours on the x-axis.
    Result: 4 lines total.
    """
    feederIcaDf = get_feederAvgIcaDf(feeder_id)
    ic_cols =          ['avg_IC10_Thermal_KW', 'avg_IC10_Voltage_KW',
                                'avg_IC90_Thermal_KW', 'avg_IC90_Voltage_KW']

    # filter to this feeder
    df_f = feederIcaDf[feederIcaDf['feeder_id'] == feeder_id].copy()
    if df_f.empty:
        print(f"No data for feeder {feeder_id}")
        return

    # make sure months are in order
      months = sorted(df_f['month'].unique()) 22
    # we will build arrays in month order
    all_hours = []
      data_by_col = {col: [] for col in ic_cols} 26
    for mi, m in enumerate(months):
      df_m = df_f[df_f['month'] == m].sort_values('hour') 29
        # x positions for this month: shift by 24 * month_index
        base = mi * 24
        x_this_month = base + df_m['hour'].to_numpy()
        all_hours.append(x_this_month)

        for col in ic_cols:
            data_by_col[col].append(df_m[col].to_numpy())

    # concatenate
    x = np.concatenate(all_hours)
    for col in ic_cols:
      data_by_col[col] = np.concatenate(data_by_col[col]) 42
    # plot
    plt.figure(figsize=(10, 5))
    for col in ic_cols:
      plt.plot(x, data_by_col[col], marker='.', label=col) 47
    # x-ticks at the start of each month block
    month_xticks = [i * 24 for i in range(len(months))]
    month_xticklabels = [f"Month {m}" for m in months]
    plt.xticks(month_xticks, month_xticklabels, rotation=0)
```

```
    plt.title(f"Feeder {feeder_id} – ICA over months (24h per month)")
    plt.xlabel("Month (each = 24 hours)") plt.ylabel("ICA (kW)")
    plt.xlim(0,                          # set xlim to start at 0
x[-1])  plot_add_grid_legend_tight()
    plt.show()

60 plot_feeder_ica_concatenated(benchmark_feeder)
```



Analysis

```
# Run functions that import raw data
evCensusGdf = import_census_block_gdf_for_charging()
evLoad2035Df = import_ev_projection_df("2035")
evLoad2025Df = import_ev_projection_df("2025")
feedersMetadataGdf = import_metaData_gdf_all_feeders()
```

```
2035
2025
```

```
1 list_some_alameda_feeders = ['013840405', '012670401', '012091102', '012501108']
2
```

Baseline

Core unmanged scenario functions

```
1
2 def get_merged_hrly_power_df(feederBaseLoadDf, feederIcaDf, evLoadDf):
3     # 1) check feeder_id uniqueness in both DFs
4     base_ids = feederBaseLoadDf["feeder_id"].unique()
ica_ids = feederIcaDf["feeder_id"].unique()
5
    if len(base_ids) != 1:
6 raise ValueError(f"feederBaseLoadDf has {len(base_ids)} feeder_ids, expected 1: {base_ids}") if len(ica_ids) != 1:
        raise ValueError(f"feederIcaDf has {len(ica_ids)} feeder_ids, expected 1: {ica_ids}")
7
8
    if (baseHour_ids[0]) != (icaHour_ids[0]):
9
        raise ValueError(f"feeder id mismatch: base={base_ids[0]}, ica={ica_ids[0]}")
1
0
    1
1
```

```
14
15      # 2) merge base + ICA, keep only one feeder_id
16          merged = pd.merge(feederIcaDf, feederBaseLoadDf, on=["month", "hour"], how="inner", suffixes=("", "_ica" 17
18      # after merge ther is 'feeder_id' (from base) and possibly 'feeder_id_ica'
19      if "feeder_id_ica" in merged.columns:
20          merged = merged.drop(columns=["feeder_id_ica"]) 21
22      # 3) repeat EV hourly data for every month present in merged
23      # evLoadDf is assumed to have columns: ['hour', ...]
24      months = merged["month"].unique()
25          months_df = pd.DataFrame({"month": months}) 26
27      # cartesian product months x evLoadDf
28          evLoad_expanded = months_df.merge(evLoadDf, how="cross") 29
                                                                                                    30      # 4) merge EV hourly data in
                                                                                  31      # now evLoad_expanded has ['month', 'hour', 'hrlyAvgEVLoadKW', 'peakHrlyEVLoadKW', ...]
                                                                                                      32      merged_all = pd.merge(
                                                                                                        33          merged,
                                                                                                    34          evLoad_expanded,
                                                                                                  35          on=["month", "hour"],
                                                                                                          36          how="left",
37      )
38
39      return merged_all
40
41  def get_results_df(mergedDf):
42      keepCols = ['feeder_id', 'division', 'month', 'hour', 'minAvg_IC_KW', 'high_base_load_kw', 'hrlyAvgEVLoa
43      resultsDf = mergedDf[keepCols].copy()
44      resultsDf['stress'] = (resultsDf['high_base_load_kw'] + resultsDf['hrlyAvgEVLoadKW']) / (resultsDf['high
45      resultsDf['overload_kw'] = resultsDf['hrlyAvgEVLoadKW'] - resultsDf['minAvg_IC_KW']
46      return resultsDf
47
48
```

```
# TESTING
# from IPython.core.interactiveshell import dis
# find all tracts intersecting the specified feeder
# Get the EV projection data for those tracts and aggregate relavent data
        # Transform ev projection data into form that matches ICA and load data, hour, month 6
avgFeederIcaDf =
get_feederAvgIcaDf(benchmark_feeder) 8
intersectingCensusBlocksDf = blocks_intersecting_feeder_df(benchmark_feeder, feedersMetadataGdf, evCensusGd
aggEvBlocksLoadGrowthDf = get_projected_evLoadGrowthDf_for_blocksDf(intersectingBlocks, evLoad2025Df,
evLoa 11
# display(aggEvBlocksLoadGrowthDf.head())

feederBaseLoadDf = get_loadDf_for_feeder(benchmark_feeder)

test_unmngdHrlyPowerDf = get_merged_hrly_power_df(feederBaseLoadDf, avgFeederIcaDf, aggEvBlocksLoadGrowthDf
test_unmngdHrlyResultsDf = get_results_df(test_unmngdHrlyPowerDf)
print('test_unmngdHrlyPowerDf')
display(test_unmngdHrlyPowerDf.head())
print('test_unmngdHrlyResultsDf')
display(test_unmngdHrlyResultsDf.head())

# Find and print the line with max overload
max_overload_row = test_unmngdHrlyResultsDf.loc[test_unmngdHrlyResultsDf['overload_kw'].idxmax()]
print("Maximum Overload:")
display(max_overload_row['overload_kw'])

# Find and print the line with max stress
max_stress_row = test_unmngdHrlyResultsDf.loc[test_unmngdHrlyResultsDf['stress'].idxmax()]
print("\nMaximum Stress:")
display(max_stress_row['stress'])
```

Found 10 blocks intersecting feeder 013840405
saved intersecting blocks test_unmngdHrlyPowerDf

	feeder_id	division	month	hour	avg_IC10_Thermal_KW	avg_IC10_Voltage_KW	avg_IC90_Thermal_KW	avg_IC90_Voltage_KW	minAvg_
0	013840405	East Bay	1	0	995.428589	768.285706	886.000000	563.714294	563.7
1	013840405	East Bay	1	1	960.571411	678.571411	842.857117	444.142853	444.1
2	013840405	East Bay	1	2	959.571411	665.285706	842.285706	435.857147	435.8
3	013840405	East Bay	1	3	959.428589	658.857117	841.857117	426.571442	426.5
4	013840405	East Bay	1	4	904.142883	527.857117	830.428589	395.571442	395.5

test_unmngdHrlyResultsDf

	feeder_id	division	month	hour	minAvg_IC_KW	high_base_load_kw	hrlyAvgEVLoadKW	stress	overload_kw
0	013840405	East Bay	1	0	563.714294	551.0	526.000001	0.966167	-37.714294
1	013840405	East Bay	1	1	444.142853	529.0	411.950005	0.966919	-32.192848
2	013840405	East Bay	1	2	435.857147	483.0	356.542012	0.913681	-79.315135
3	013840405	East Bay	1	3	426.571442	450.0	278.056006	0.830572	-148.515435
4	013840405	East Bay	1	4	395.571442	420.0	202.512003	0.763283	-193.059438

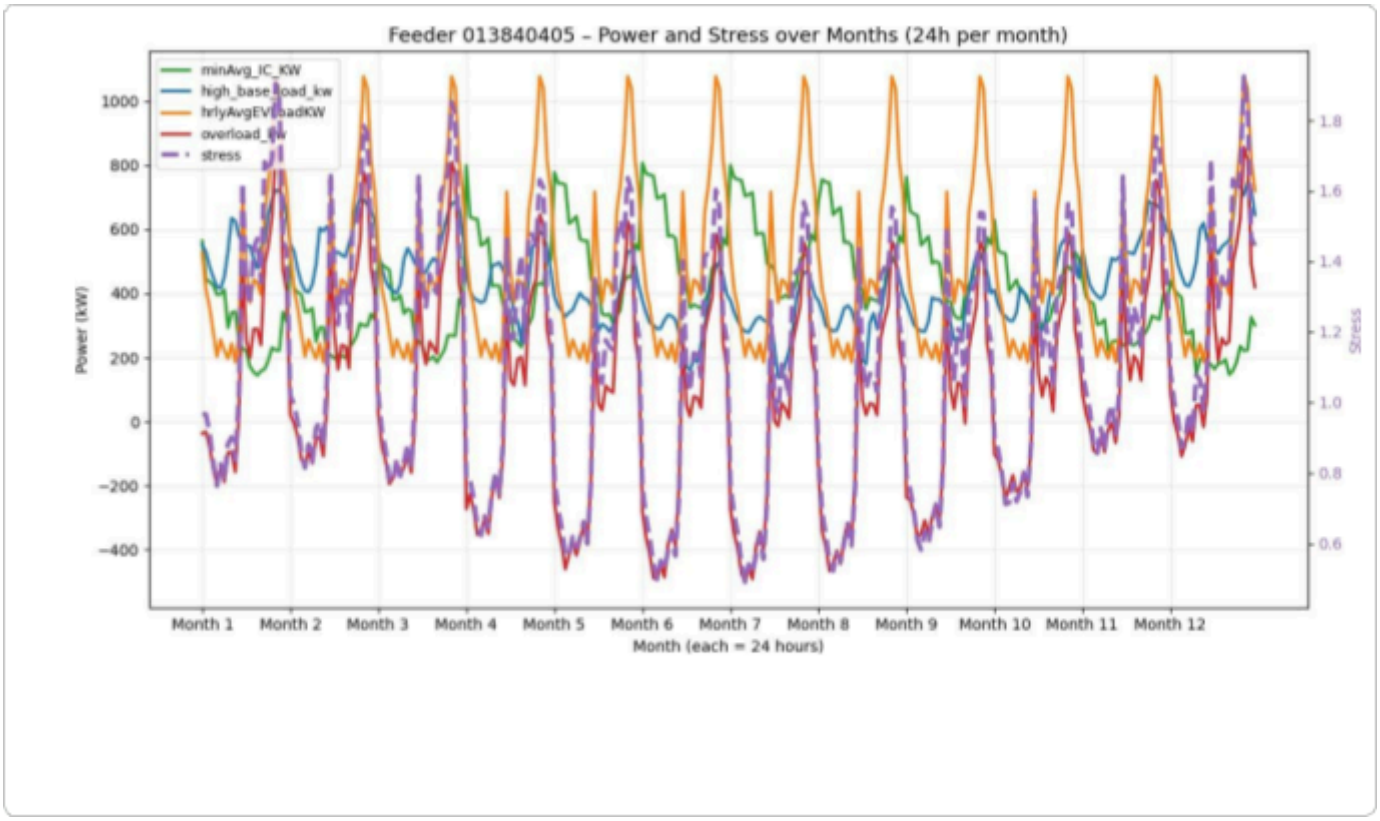
Maximum Overload:
np.float64(856.6548673942693)

Maximum Stress:
np.float64(1.9242577132481162)

葙 Plots for Unmanged Results

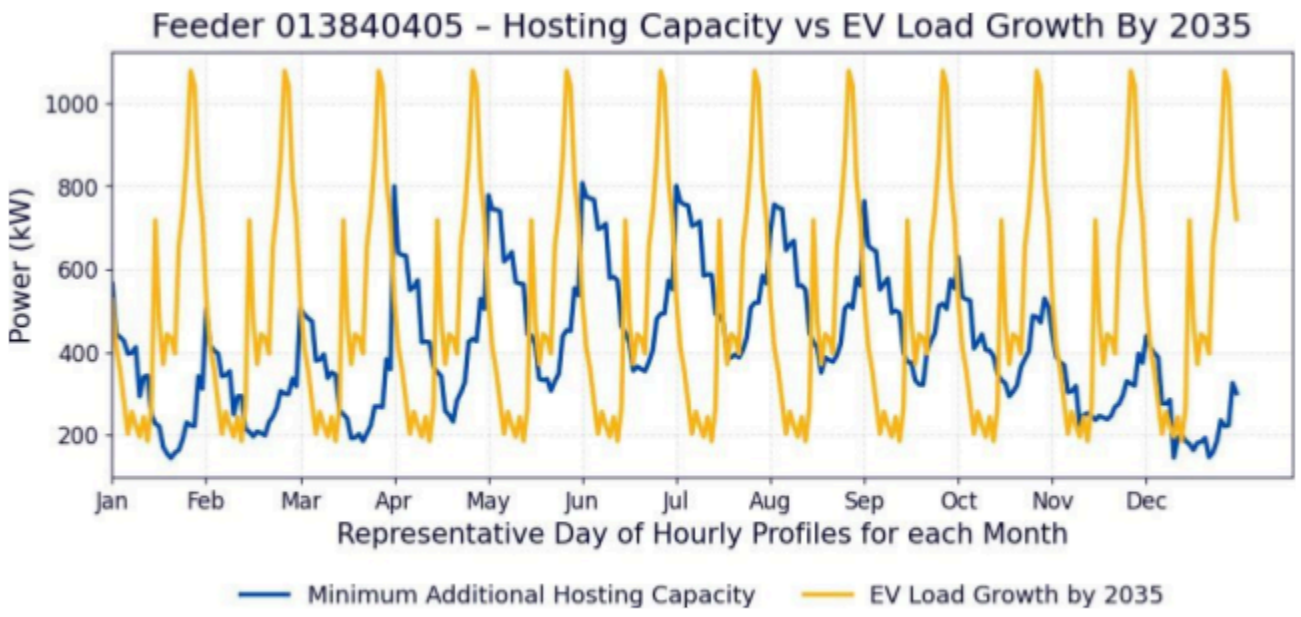
```
def plot_feeder_results(resultsDf, feeder_id_col="feeder_id"):
"""
Plot power (kW) and stress for a single feeder across all months and hours.
resultsDf must include columns:
```

```
                                                    ['month', 'hour',
                                                     'stress',
    """                                              'minAvg_IC_KW',
                                                     'high_base_load_kw',
                                                     'hrlyAvgEVLoadKW',
                                                     'overload_kw']
    # --- Ensure sorted by month/hour ---
      resultsDf = resultsDf.sort_values(["month", "hour"]).reset_index(drop=True) 11
    # --- Build x-axis (1–288 points for 12×24) ---
    x = range(len(resultsDf))
    xticks = [i * 24 for i in range(resultsDf["month"].nunique())]
      xlabels = [f"Month {m}" for m in sorted(resultsDf["month"].unique())] 16
    # --- Create figure with twin y-axes ---
    fig, ax1 = plt.subplots(figsize=(12, 6))

    # --- Plot power data (left y-axis, in kW) ---
    power_cols = ["minAvg_IC_KW", "high_base_load_kw", "hrlyAvgEVLoadKW", "overload_kw"]
    colors = ["tab:green", "tab:blue", "tab:orange", "tab:red"]
    for col, color in zip(power_cols, colors):
          if col in resultsDf.columns:
                ax1.plot(x, resultsDf[col], label=col, color=color, linewidth=2)
      # ax1.scatter(x, resultsDf[col], color=color, s=10) 27
    ax1.set_xlabel("Month (each = 24 hours)")
    ax1.set_ylabel("Power (kW)", color="black")
    ax1.tick_params(axis="y", labelcolor="black")
    ax1.set_xticks(xticks)
    ax1.set_xticklabels(xlabels, rotation=0)
    ax1.grid(True, alpha=0.3)

    # --- Secondary y-axis for stress ---
    ax2 = ax1.twinx()
    ax2.plot(x, resultsDf["stress"], label="stress", color="tab:purple", linewidth=3, linestyle="--")
    ax2.set_ylabel("Stress", color="tab:purple")
    ax2.tick_params(axis="y", labelcolor="tab:purple")

    # --- Combine legends from both axes ---
    lines, labels = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
      ax1.legend(lines + lines2, labels + labels2, loc="upper left", fontsize=9) 45
    # --- Title ---
    feeder_id = resultsDf[feeder_id_col].iloc[0] if feeder_id_col in resultsDf.columns else "Feeder"
      ax1.set_title(f"Feeder {feeder_id} – Power and Stress over Months (24h per month)", fontsize=13) 49
    plt.tight_layout()
    plt.show()

plot_feeder_results(test_unmngdHrlyResultsDf)
```




```
# PRETTY PLOT FOR PRESENTATIONS
def plot_feeder_results_pretty(
    resultsDf,
    power_cols=["minAvg_IC_KW", "hrlyAvgEVLoadKW"],
    legend_labels=["Minimum Additional Hosting Capacity","EV Load Growth by 2035"],
    title=None,
    legend_title="Legend", feeder_id_col="feeder_id",
    """
    Simpler, presentation-style plot:
    - Plots selected power columns (default: minAvg_IC_KW, hrlyAvgEVLoadKW)
    - Uses nicer colors and legend labels
    """
):
    # --- Ensure sorted by month/hour ---
    resultsDf = resultsDf.sort_values(["month", "hour"]).reset_index(drop=True)

    # --- Build x-axis (1–288 points for 12×24) ---
    x = range(len(resultsDf))
    months = sorted(resultsDf["month"].unique())
    xticks = [resultsDf[resultsDf["month"] == m].index[0] for m in months]
    month_xticklabels = [calendar.month_abbr[m] for m in months] # <-- NEW
    # xlabels = [f"Month {m}" for m in months] #

    # --- Create figure ---
    fig, ax = plt.subplots(figsize=(10, 5))

    # --- Plot requested power columns ---
    for i, col in enumerate(power_cols):
        if col not in resultsDf.columns: continue # skip if missing
            label = legend_labels[i] if i < len(legend_labels) else col
            color = LINE_COLORS[i % len(LINE_COLORS)]
        ax.plot(x, resultsDf[col], label=label, color=color, linewidth=2.5)
    # --- X-axis formatting ---
```

```
    ax.set_xticks(xticks)
    ax.set_xticklabels(month_xticklabels, rotation=0)
    ax.set_xlim(xmin=0)

    # --- Apply shared style ---
    feeder_id = (
        resultsDf[feeder_id_col].iloc[0]
        if feeder_id_col in resultsDf.columns
        else "Feeder"
    )
    if title is None:
        title = f"Feeder {feeder_id} – Hosting Capacity vs EV Load Growth By 2035"
    apply_pretty_plot_style(
        ax,
        title=title,
        xlabel="Representative Day of Hourly Profiles for each Month",
        ylabel="Power (kW)",
    )

    # --- Legend styling ---
    legend = ax.legend(frameon=False, ncol=3, loc="upper center", bbox_to_anchor=(0.5, -0.2), fontsize=13)
    for text in legend.get_texts():
        text.set_color(MAIN_COLOR)

    plt.tight_layout()
    plt.show()
plot_feeder_results_pretty(test_unmngdHrlyResultsDf)
```



葹 Run Unmanaged Charging

```
1
2 # find all tracts intersecting the specified feeder
3 # Get the EV projection data for those tracts and aggregate relavent data
4             # Transform ev projection data into form that matches ICA and load data, hour, month 5
6 def run_unmanaged_scenario(feeder_id, evCensusGdf=evCensusGdf, evLoadNowDf=evLoad2025Df, evLoadFutureDf=evL
7         benchmark_feeder = feeder_id
8         feederBaseLoadDf = get_loadDf_for_feeder(benchmark_feeder)
9         avgFeederIcaDf = get_feederAvgIcaDf(benchmark_feeder)
10        intersectingCensusBlocksDf = blocks_intersecting_feeder_df(benchmark_feeder, feedersMetadataGdf, evCens
11        # aggEvLoadDf = get_aggregate_evLoadDf_for_blocksDf(intersectingBlocksCensusDf, projectedEvLoadDf)
12        aggEvBlocksLoadGrowthDf = get_projected_evLoadGrowthDf_for_blocksDf(intersectingCensusBlocksDf, evLoad2
13           hrlyPowerDf = get_merged_hrly_power_df(feederBaseLoadDf, avgFeederIcaDf, aggEvBlocksLoadGrowthDf) 14
15        hrlyResultsDf = get_results_df(hrlyPowerDf)
16           return hrlyPowerDf, hrlyResultsDf 17
18
19 def plot_and_print_unmanaged_results(feeder_id, df_hrlyResults):
20        print(f"Feeder {feeder_id} Unmanaged Results:")
21        # display(df_hrlyResults)
22
23        # Find and print the line with max overload
24        max_overload_row = df_hrlyResults.loc[df_hrlyResults['overload_kw'].idxmax()]
25        print("Maximum Overload:")
26        display(max_overload_row['overload_kw'])
27
                                                                                    28          # Find and print the line with max stress
                                                                        29          max_stress_row = df_hrlyResults.loc[df_hrlyResults['stress'].idxmax()]
                                                                                    30          print("\nMaximum Stress:")
                                                                                31          display(max_stress_row['stress'])
32
33        # plot_feeder_results(df_hrlyResults)
34        plot_feeder_results_pretty(df_hrlyResults)
35        # interactive_results_plot(df_hrlyResults)
36
37
38
```

```
"   for feeder in list_some_alameda_feeders:
"   hrlyPowerDf, hrlyResultsDf = run_unmanaged_scenario(feeder)
"   plot_and_print_unmanaged_results(feeder, hrlyResultsDf)
                4
```

Show hidden output

Double-click (or enter) to edit

Double-click (or enter) to edit

葹 Managed Charging

```
# Global parameters for managed charging
P_max_chg = 7.0 # kW per charger
managed_share= 0.8
stress_weight_exp=3.0 # >1 => put even more weight on low-stress hours
```

Scenario 1: LINEAR charging curve

7 cells hidden

葹 Court Modified Linear Charging Curve

```
def get_total_number_of_chargers(hrlyPowerDf, P_max=P_max_chg):
    peak_ev = hrlyPowerDf['hrlyAvgEVLoadKW'].max()
    N_chargers_tot = int(np.ceil(peak_ev / P_max)) if peak_ev > 0 else 0
    return N_chargers_tot

def attach_unmanaged_stress_from_results(hrlyPowerDf, hrlyResultsDf):
    """
    Use the 'stress' column from hrlyResultsDf (unmanaged scenario)
  as the unmanaged stress for hrlyPowerDf. 10
    Assumes hrlyPowerDf and hrlyResultsDf are aligned row-by-row
    (same feeder, same hours in the same order).
    """
    df = hrlyPowerDf.copy()
    df['stress_unmanaged'] = hrlyResultsDf['stress'].values
    return df

def split_ev_load_managed_unmanaged(df, managed_share=managed_share):
    """
    Split aggregate EV load into:
        - ev_unmanaged_part_kw = (1 - f) * EV_unmanaged
        - ev_managed0_kw               = f               * EV_unmanaged
    where f = managed_share.
    """
    df = df.copy()
    f = float(managed_share)
    df['ev_unmanaged_part_kw'] = (1.0 - f) * df['hrlyAvgEVLoadKW']
    df['ev_managed_kw']                  = f                * df['hrlyAvgEVLoadKW']
    return df

def compute_stress_based_weights(df_splitEVLoad, stress_weight_exp=stress_weight_exp):
    """
    Compute normalized weights w_norm(t) based on unmanaged stress:
        w_raw(t) = 1 / (1 + Stress^u(t)^exp)
      w_norm(t) = w_raw(t) / C 36
    Constant C is chosen so that:
        sum_t w_norm(t) * ev_managed0_kw(t) = sum_t ev_managed0_kw(t)
    (no energy loss on the managed part).
    """
      df = df_splitEVLoad.copy() 42
    df['w_raw'] = 1.0 / (1.0 + df['stress_unmanaged']**stress_weight_exp)

    num = (df['w_raw'] * df['ev_managed_kw']).sum()
    den = df['ev_managed_kw'].sum()
      C = num / den if den > 0 else 1.0 48
    df['w_norm'] = df['w_raw'] / C
    return df

def apply_weights_to_managed_ev(df):
    """
    Use w_norm to create:
```

- ev_managed_kw = w_norm * ev_managed0_kw
- hrlyManagedEVLoadKW = ev_unmanaged_part_kw + ev_managed_kw

```
def build_managed_profile_for_feeder(df_hrlyPowerData, df_unmangedHrlyResults):
    """
    Fully managed EV charging profile pipeline.
    All external inputs are passed in.
    Only global constants used:
    - P_max_chg
    - managed_share
    - stress_weight_exp
    """

    df = df_hrlyPowerData.copy()

    # 1) Attach unmanaged stress
    df = attach_unmanaged_stress_from_results(df, df_unmangedHrlyResults)

    # 2) Compute charger counts
    N_ch_tot = get_total_number_of_chargers(df, P_max_chg)
    N_ch_managed = int(round(managed_share * N_ch_tot))
    df['N_ch_tot'] = N_ch_tot
    df['N_ch_managed'] = N_ch_managed

    # 3) Split EV load
    df = split_ev_load_managed_unmanaged(df, managed_share)

    # 4) Compute stress-based weights
    df = compute_stress_based_weights(df, stress_weight_exp)

    # 5) Apply weights to EV managed load
    df_managedResults = apply_weights_to_managed_ev(df)

    # cleanup
    df_managedResults = df_managedResults.drop(columns=['w_raw', 'w_norm'], errors='ignore')
    return df_managedResults

def get_managed_results_df(mergedDf):
    """

    """
    keepCols = ['feeder_id', 'division', 'month', 'hour', 'minAvg_IC_KW', 'high_base_load_kw', 'hrlyManaged
    resultsDf = mergedDf[keepCols].copy()
    resultsDf['managed_stress'] = (resultsDf['high_base_load_kw'] + resultsDf['hrlyManagedEVLoadKW']) / (re
    resultsDf['managed_overload_kw'] = resultsDf['hrlyManagedEVLoadKW'] - resultsDf['minAvg_IC_KW']
    return resultsDf
```

```
# TESTING

# 1) Build managed hourly power DF (with hrlyManagedEVLoadKW)

test_mngdHrlyPowerDf = build_managed_profile_for_feeder(test_unmngdHrlyPowerDf, test_unmngdHrlyResultsDf)
# 2) Recompute stress + overload using the NEW EV load

res_unmg = test_unmngdHrlyResultsDf.copy()

res_mng = get_managed_results_df(test_mngdHrlyPowerDf)


fused_stress_df = pd.merge(
res_mng,
res_unmg,
on=['feeder_id', 'division', 'month', 'hour', 'minAvg_IC_KW', 'high_base_load_kw'],
how='inner'
```

```
        17 )
display(res_mng.head())
display(fused_stress_df.head())
```

	feeder_id	division	month	hour	minAvg_IC_KW	high_base_load_kw	hrlyManagedEVLoadKW	N_ch_tot	N_ch_managed	managed_stres	
1	013840405	East Bay	1	1	444.142853	29.0	44.946918	54	23	1.10358	1
0	013840405	East Bay	1	0	563.714294	51.0	96.471976	54	23	1.11909	1
2	013840405	East Bay	1	2	435.857147	483.0	503.730423	154	123	1.07386	
3	013840405	East Bay	1	3	426.571442	450.0	433.531252	154	123	1.11909	
4	013840405	East Bay	1	4	395.571442	20.0	40.187019	54	23	1.00794	0

	feeder_id	division	month	hour	minAvg_IC_KW	high_base_load_kw	hrlyManagedEVLoadKW	N_ch_tot	N_ch_managed	managed_stres	
0	013840405	East Bay	1	0	563.714294	51.0	96.471976	54	23	1.11909	1
1	013840405	East Bay	1	1	444.142853	529.0	544.946918	154	123	1.10358	
2	013840405	East Bay	1	2	435.857147	483.0	503.730423	154	123	1.07386	
3	013840405	East Bay	1	3	426.571442	50.0	33.531252	54	23	1.00794	1
4	013840405	East Bay	1	4	395.571442	420.0	340.187019	154	123	0.93209	

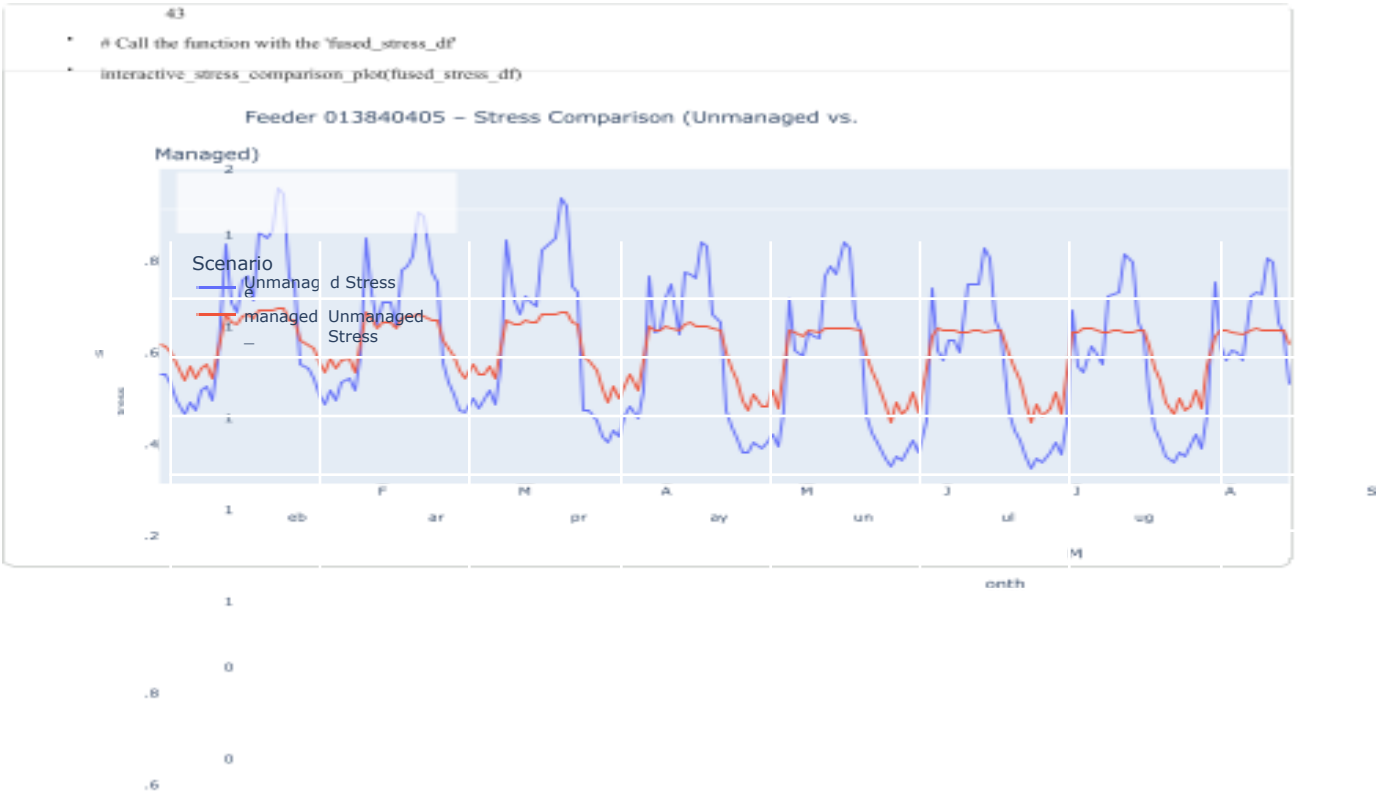
葎 Plots for Managed

```
def plot_ev_load_comparison(feeder_id, hrlyPowerDf_unmanaged, managedEvProfileDf):
    """
    Plots hrlyAvgEVLoadKW (unmanaged) and hrlyManagedEVLoadKW against time,
concatenating months and hours.
    """
    # Filter data for the specific feeder
    df_unmanaged_f = hrlyPowerDf_unmanaged[hrlyPowerDf_unmanaged['feeder_id'] == feeder_id].copy()
df_managed_f = managedEvProfileDf[managedEvProfileDf['feeder_id'] == feeder_id].copy()

if df_unmanaged_f.empty or df_managed_f.empty: print(f"No data
 for feeder {feeder_id}") return

    # Merge the relevant columns
merged_ev_df = pd.merge(
        df_unmanaged_f[['feeder_id', 'month', 'hour', 'hrlyAvgEVLoadKW']], df_managed_f[['feeder_id',
 'month', 'hour', 'hrlyManagedEVLoadKW']], on=['feeder_id', 'month', 'hour'],
        how='inner'

    )

    # Ensure sorted by month and hour

    merged_ev_df = merged_ev_df.sort_values(by=['month', 'hour']).reset_index(drop=True)

    # Prepare for concatenation on x-axis

    months = sorted(merged_ev_df['month'].unique())
all_hours_x = []
    hrly_avg_ev_load = []
hrly_managed_ev_load = []

    for mi, m in enumerate(months):

        df_m = merged_ev_df[merged_ev_df['month'] == m].sort_values('hour')


        base = mi * 24
```

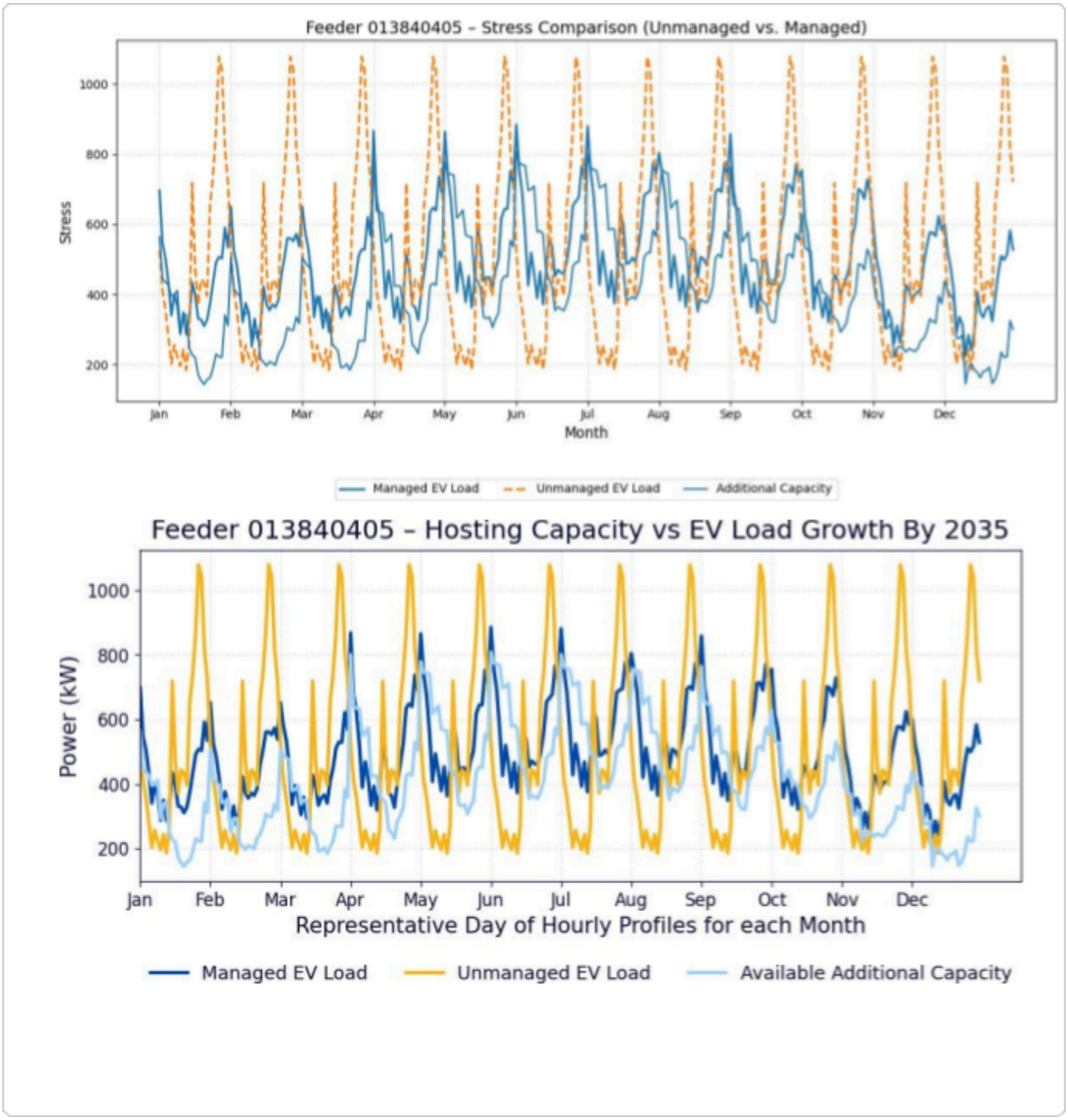
```
    y_avg = np.concatenate(hrly_avg_ev_load)
    y_managed = np.concatenate(hrly_managed_ev_load)

    # Plotting plt.figure(figsize=(10, 5))
    plt.plot(x, y_avg, marker='.', label='Unmanaged hrlyAvgEVLoadKW', color='tab:blue') plt.plot(x, y_managed, marker='.', label='Managed hrlyManagedEVLoadKW', color='tab:orange')

    # X-ticks for months
    month_xticks = [i * 24 for i in range(len(months))] month_xticklabels = [f"Month {m}" for m in months] plt.xticks(month_xticks, month_xticklabels, rotation=0)

    plt.title(f"Feeder {feeder_id} – EV Load Comparison (Unmanaged vs. Managed)") plt.xlabel("Month (each = 24 hours)")
# Call the plotting function for the benchmark feeder
plot_ev_load_comparison(benchmark_feeder, test_unmngdHrlyPowerDf, res_mng)
    plt.xlim(0, x[-1])
    plot_add_grid_legend_right() plt.show()
```

Show hidden output

```
def interactive_stress_comparison_plot(fused_df):
    # Ensure the DataFrame is sorted by month and hour for correct timeline representation
    fused_df = fused_df.sort_values(by=['month', 'hour']).reset_index(drop=True)
    # Calculate x-axis tick positions and labels for months
    months = sorted(fused_df['month'].unique())
    month_xticks = [fused_df[fused_df['month'] == m].index[0] for m in months]
    month_xticklabels = [calendar.month_abbr[m] for m in months]
    # Use plotly.express.line to generate the plot, plotting 'stress' and 'managed_stress' directly
    fig = px.line(
        fused_df,
        x=fused_df.index,
        y= ['stress', 'managed_stress'], # Plot both stress columns directly
      hover_data=["month", "hour"] , #"stress", "managed_stress"], # Configure hover_data
        labels={
            "value": "Stress",
            "variable": "Scenario",
            "index": "Time (Month-Hour)"
        }, # Set appropriate labels
        title=f"Feeder {fused_df['feeder_id'].iloc[0]} – Stress Comparison (Unmanaged vs. Managed)" # Set a
    )

    # Customize legend names for clarity
    fig.for_each_trace(lambda t: t.update(name = t.name.replace('stress', 'Unmanaged Stress').replace('mana
    # Update the layout to specify y-axis titles and legend position and x-axis ticks
    fig.update_layout(
        yaxis_title="Stress",
        legend=dict(x=0.01, y=0.99, bgcolor='rgba(255,255,255,0.7)', bordercolor='rgba(0,0,0,0.5)')
    )

    fig.update_xaxes(
        tickmode='array',
        tickvals=month_xticks,
        ticktext=month_xticklabels,
        title_text="Month"
    )

    # Display the plot
    fig.show()
```




```
def plot_stress_comparison_matplotlib(fused_df):
    # Ensure the DataFrame is sorted by month and hour for correct timeline representation
        fused_df = fused_df.sort_values(by=['month', 'hour']).reset_index(drop=True) 4
    # Calculate x-axis tick positions and labels for months
    months = sorted(fused_df['month'].unique())
    month_xticks = [fused_df[fused_df['month'] == m].index[0] for m in months]
        month_xticklabels = [calendar.month_abbr[m] for m in months] 9
    plt.figure(figsize=(12, 6))

    # plt.plot(fused_df.index, fused_df['stress'], label='Unmanaged Stress', color='tab:blue', linewidth=2)
    # plt.plot(fused_df.index, fused_df['managed_stress'], label='Managed Stress', color='tab:orange', line
    plt.plot(fused_df.index, fused_df['hrlyManagedEVLoadKW'], label='Managed EV Load', color='tab:blue', li
    plt.plot(fused_df.index, fused_df['hrlyAvgEVLoadKW'], label='Unmanaged EV Load', color='tab:orange', li
       plt.plot(fused_df.index, fused_df['minAvg_IC_KW'], label ='Additional Capacity') 17
    plt.title(f"Feeder {fused_df['feeder_id'].iloc[0]} – Stress Comparison (Unmanaged vs. Managed)", fontsi
    plt.xlabel("Month", fontsize=12)
    plt.ylabel("Stress", fontsize=12)

    plt.xticks(month_xticks, month_xticklabels, rotation=0, fontsize=10)
    plt.grid(True, linestyle=':', alpha=0.7)
    plt.legend(fontsize=10, loc='upper center', bbox_to_anchor=(0.5, -0.2), ncol=3)
    plt.tight_layout()
    plt.show()

# Call the Matplotlib plotting function with the 'fused_stress_df'
plot_stress_comparison_matplotlib(fused_stress_df)

plot_feeder_results_pretty(
    fused_stress_df,
    power_cols=['hrlyManagedEVLoadKW', 'hrlyAvgEVLoadKW', 'minAvg_IC_KW'],
    legend_labels=["Managed EV Load","Unmanaged EV Load", "Available Additional Capacity"],
    )
```



蕙 Run

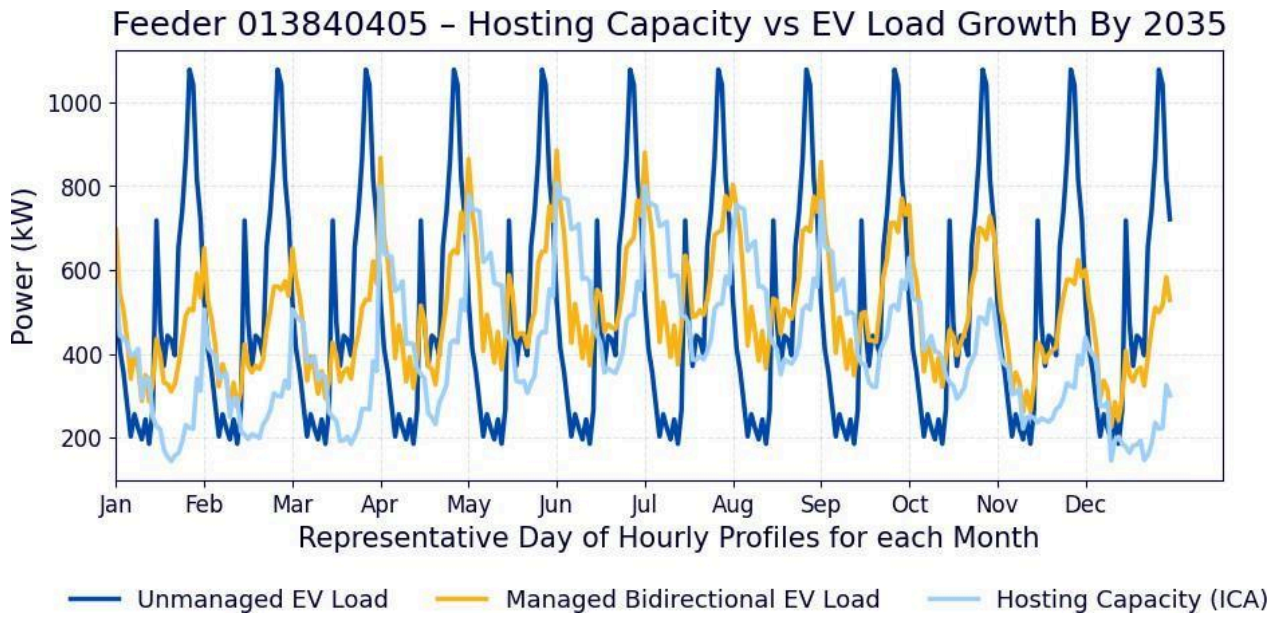
```
# Function to RUN BASELINE & UNMANAGED
def run_[illegible](...):
    df_by_Power, df_unmanaged_results = run_unmanaged_scenario(feeder_id)

    # Managed Scenario
    df_managed_by_Power = build_managed_profile_for_feeder(df_by_Power, df_unmanaged_results)

    df_managed_results = get_managed_results_df(df_managed_by_Power)
    display(df_managed_by_Power.head())
    # display(df_managed_results.head()); print("...")
    pd.Series(
        df_unmanaged_results.copy(),
        df_managed_results.copy(),
    )
```

```
                on=['feeder_id', 'division', 'month', 'hour', 'minAvg_IC_KW', 'high_base_load_kw'],
                how="inner"
            )

            plot_feeder_results_pretty(
                fused_stress_df,

                power_cols=['hrlyAvgEVLoadKW', 'hrlyManagedEVLoadKW', 'minAvg_IC_KW'], legend_labels=["Unmanaged
    EV Load", "Managed Bidirectional EV Load","Hosting Capacity (ICA)"], # legend_title = ""
                )


            # plot_and_print_unmanaged_results(feeder_id, df_unmanagedHrlResults) display(fused_stress_df.head())

            return fused_stress_df

# # def plot_and_print_unmanaged_and_managed_results():
# run_managed_and_benchmark_scenario("01138440405")
```

```
# Function to get all info for cost analysis as dataframe and save
def get_cost_input_df(df_fusedStress, costInputDf=None):
    # Peak Overload both cases
    # Feeder annual energy delivered
    df = pd.DataFrame({
        "feeder_id": [df_fusedStress["feeder_id"].unique()[0]],
        "totYrlyFeederEnergy_Managed_kWh": (df_fusedStress["high_base_load_kw"]
                                            + df_fusedStress["hrlyManagedEVLoadKW"]).sum(),

        "totYrlyFeederEnergy_Unmanaged_kWh": (df_fusedStress["high_base_load_kw"]
                                              + df_fusedStress["hrlyAvgEVLoadKW"]).sum(),

        "PeakOverload_Manageged_kW": df_fusedStress["managed_overload_kw"].max(),

        "PeakOverload_Unmanaged_kW": df_fusedStress["overload_kw"].max(),

        "MaxStress_Unmanaged": df_fusedStress["stress"].max(),

        "MaxStress_Managed": df_fusedStress["managed_stress"].max(),

        "Num_ManagedBiChargers": df_fusedStress["N_ch_managed"].unique(), # each row should have same value

        "Num_UnmanagedBiChargers": df_fusedStress["N_ch_tot"].unique() # each row should have same value ch
        })
    # If costInputDf
    if costInputDf is None:
        costInputDf = df
    else:
        costInputDf = pd.concat([costInputDf, df], ignore_index=True)
    return costInputDf


def save_cost_input_df(df_fusedStress):
    # get feeder id from df
    feeder_id = df_fusedStress["feeder_id"].unique()[0]
    # name output file based on feeder idf
    output_file = f"costInputsFromGridStressAnalysis_Feeder_{feeder_id}.csv"
    save_path = os.path.join(court_base_path, "Cost_Input_from_Grid_Stress_Analysis", output_file)

    costInputDf.to_csv(save_path, index=False)

    print("Saved to:", save_path)
```

```
# fusedStress = run_managed_and_benchmark_scenario('013840405')
# costInputDf = get_cost_input_df(fusedStress)


# save_cost_input_df(fusedStress)
# display(costInputDf)

# # save CostInputDF as 'costInputsFromGridStressAnalysis_Feeder_013840405.csv' in current directory
        # costInputDf.to_csv('costInputsFromGridStressAnalysis_Feeder_013840405.csv', index=False) 8

for feeder in list_some_alameda_feeders:
    fusedStress = run_managed_and_benchmark_scenario(feeder)
    display(fusedStress.head())
    # print(fusedStress['feeder_id'].unique())
    costInputDf = get_cost_input_df(fusedStress)
    display(costInputDf)
    save_cost_input_df(fusedStress)
```

```
Found 10 blocks intersecting feeder 013840405
saved intersecting blocks
```



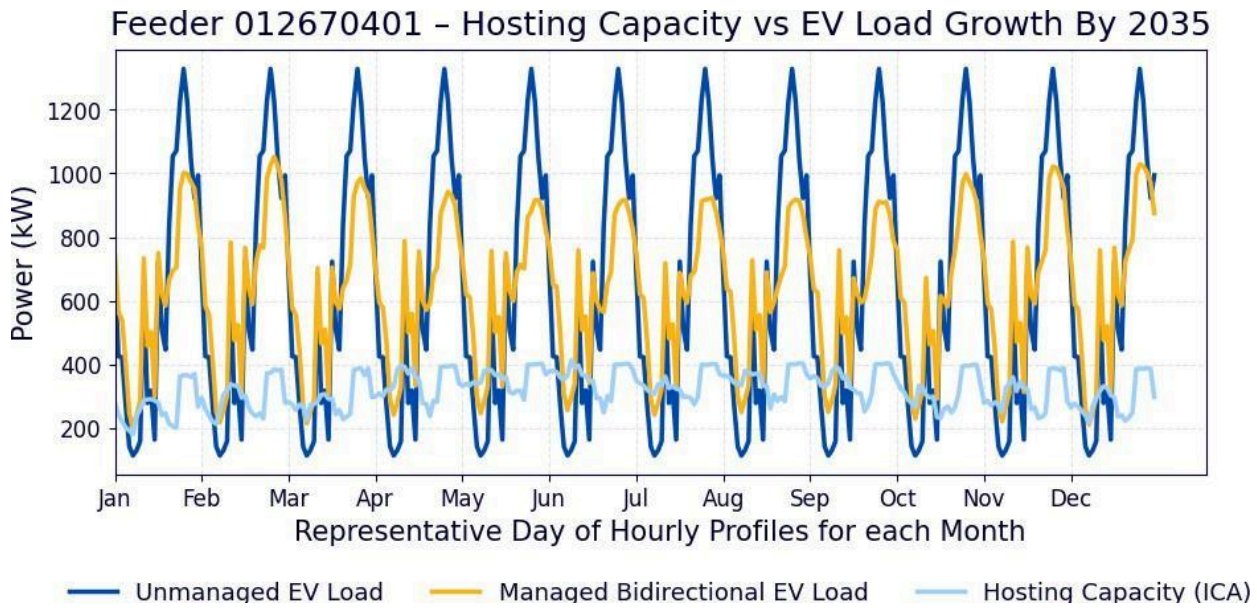
	feeder_id	division	month	hour	minAvg_IC_KW	high_base_load_kw	hrlyAvgEVLoadKW	stress	overload_kw	hrlyManagedEVLoadKW
0	013840405	East Bay	1	0	563.714294	551.0	526.000001	0.966167	-37.714294	696.471976
1	013840405	East Bay	1	1	444.142853	529.0	411.950005	0.966919	-32.192848	544.946918
2	013840405	East Bay	1	2	435.857147	483.0	356.542012	0.913681	-79.315135	503.730423
3	013840405	East Bay	1	3	426.571442	450.0	278.056006	0.830572	-148.515435	433.531252
4	013840405	East Bay	1	4	395.571442	420.0	202.512003	0.763283	-193.059438	340.187019

	feeder_id	division	month	hour	minAvg_IC_KW	high_base_load_kw	hrlyAvgEVLoadKW	stress	overload_kw	hrlyManagedEVLoadKW
0	013840405	East Bay	1	0	563.714294	551.0	526.000001	0.966167	-37.714294	696.471976
1	013840405	East Bay	1	1	444.142853	529.0	411.950005	0.966919	-32.192848	544.946918
2	013840405	East Bay	1	2	435.857147	483.0	356.542012	0.913681	-79.315135	503.730423
3	013840405	East Bay	1	3	426.571442	450.0	278.056006	0.830572	-148.515435	433.531252
4	013840405	East Bay	1	4	395.571442	420.0	202.512003	0.763283	-193.059438	340.187019

	feeder_id	totYrlyFeederEnergy_Managed_kWh	totYrlyFeederEnergy_Unmanaged_kWh	PeakOverload_Manageged_kW	PeakOverload_Unmana

0	013840405	270312.292909	270312.292909	292.001862	856

```
Saved to: /content/drive/MyDrive/Courses/CE256 Sust Trans/[CE 256] Grid Impact Group Project/Grid Stress Analysis/Cost_Input_fro
Found 16 blocks intersecting feeder 012670401
saved intersecting blocks
```



Appendix D: Code – Cost Analysis

```
import pandas as pd
import os

# Cost per charger (you can adjust this if needed)
CHARGER_INSTALL_COST = 5500.0 # $ per charger

# PG&E DDOR Table 3 - Circuit upgrade costs only (MEDIAN, $/kW)
# Interpreted as marginal $/kW within each band (tiered pricing).
# Bands are defined in kW of GRID NEED (i.e., overload), not total load.
CIRCUIT_TIERED_RATES_MEDIAN = [
(0.0, 1000.0, 1875.00), # <1 MW
(1000.0, 2000.0, 1368.89), # ≥1 & <2 MW
(2000.0, 4000.0, 673.35), # ≥2 & <4 MW
(4000.0, 8000.0, 438.14), # ≥4 & <8 MW
(8000.0, float("inf"), 367.85), # ≥8 MW
]



def calculate_upgrade_cost(overload_kw: float) -> float:
"""
Tiered circuit upgrade cost using MEDIAN circuit $/kW only.
```

```
    We treat each band as a marginal cost tier (like tax brackets):
    - First 0–1 MW at 1875 $/kW
    - Next 1–2 MW at 1368.89 $/kW
    - Next 2–4 MW at 673.35 $/kW
    - Next 4–8 MW at 438.14 $/kW
    - Anything above 8 MW at 367.85 $/kW

    This ensures:
    - Cost is 0 if overload ≤ 0
    - Total cost is non-decreasing in overload_kw
    """
    if overload_kw is None:
    return 0.0

    try:
    overload_kw = float(overload_kw)
    except (TypeError, ValueError):
    return 0.0

    if overload_kw <= 0:
    return 0.0

    cost = 0.0
    for lower_kw, upper_kw, rate_per_kw in CIRCUIT_TIERED_RATES_MEDIAN:
    if overload_kw > lower_kw:
    # Amount of kW that actually falls inside this tier
    kw_in_tier = min(overload_kw, upper_kw) - lower_kw
    cost += kw_in_tier * rate_per_kw
    else:
    # Overload doesn't reach this tier; we can stop
    break

    return cost



    def process_csv(file_path, output_file=None):
    """
    Process a combined CSV file where each row is a feeder.

    For each row (feeder_id), calculate:
    - Managed_Upgrade_Cost
    - Charger_Cost
    - Unmanaged_Upgrade_Cost
    - Total_Managed_Cost
    - Savings

    And append these as new columns to the existing dataframe.
    """
    try:
    # Read the CSV file
```

```
df = pd.read_csv(file_path)

# Optional: make feeder_id the index if you like
# (not required for the calculations to work)
if 'feeder_id' in df.columns:
df.set_index('feeder_id', inplace=True)

# Calculate costs per row
managed_upgrade_costs = []
unmanaged_upgrade_costs = []
charger_costs = []
total_managed_costs = []
savings_list = []

for idx, row in df.iterrows():
# Managed scenario overload (kW)
managed_overload = row.get('PeakOverload_Manageged_kW', 0.0)
managed_upgrade = calculate_upgrade_cost(managed_overload)

# Number of managed bidirectional chargers
num_chargers = row.get('Num_ManagedBiChargers', 0.0)
try:
num_chargers = float(num_chargers)
except (TypeError, ValueError):
num_chargers = 0.0

charger_cost = num_chargers * CHARGER_INSTALL_COST

# Unmanaged scenario overload (kW)
unmanaged_overload = row.get('PeakOverload_Unmanaged_kW', 0.0)
unmanaged_upgrade = calculate_upgrade_cost(unmanaged_overload)

# Totals & savings
total_managed = managed_upgrade + charger_cost
savings = unmanaged_upgrade - total_managed

managed_upgrade_costs.append(managed_upgrade)
unmanaged_upgrade_costs.append(unmanaged_upgrade)
charger_costs.append(charger_cost)
total_managed_costs.append(total_managed)
savings_list.append(savings)

# Append new columns to the existing dataframe
df['Managed_Upgrade_Cost'] = managed_upgrade_costs
df['Charger_Cost'] = charger_costs
df['Unmanaged_Upgrade_Cost'] = unmanaged_upgrade_costs
df['Total_Managed_Cost'] = total_managed_costs
df['Savings'] = savings_list

# Reset index so feeder_id is a normal column again (if you prefer)
```

```
        df.reset_index(inplace=True)

        # Build default output filename if not provided
        if output_file is None:
        base, ext = os.path.splitext(file_path)
        output_file = f"{base}_with_costs{ext}"

        # Save to CSV
        df.to_csv(output_file, index=False)
        print(f"\nResults saved to {os.path.abspath(output_file)}")

        # Summary totals
        total_managed = df['Total_Managed_Cost'].sum()
        total_unmanaged = df['Unmanaged_Upgrade_Cost'].sum()
        total_savings = df['Savings'].sum()

        print("\nSummary (all feeders in combined file):")
        print(f"Total Managed Scenario Cost: ${total_managed:,.2f}")
        print(f"Total Unmanaged Scenario Cost: ${total_unmanaged:,.2f}")
        print(f"Total Savings with Managed Scenario: ${total_savings:,.2f}")

        return df

        except Exception as e:
        print(f"Error processing file: {e}")
        return None



        if __name__ == "__main__":
        # Combined input file with all feeders
        input_csv = "cost_inputs_combined.csv"

        process_csv(input_csv)
```