M3 Challenge 2025:

Hot Button Issue: *Staying Cool as the World Heats Up*

Team #18076

March 1st, 2025

Executive Summary

To the City Council of Memphis,

As global warming intensifies, heat waves are increasingly frequent, severe, and prolonged, posing significant risks to urban populations [1]. In Memphis, these events strain the electrical grid, leading to power outages that exacerbate the dangers of extreme heat, particularly for socioeconomically vulnerable communities. In order to address this, our team developed mathematical models to predict the indoor temperatures in non-air-conditioned homes, forecast peak power demand, and assess neighborhood vulnerability, equipping your city with the information to mitigate these risks effectively.

For predicting indoor temperatures during a heat wave, we utilized a dynamic thermal network model, applying Memphis-specific dwelling data and specialized physics principles to account for heat transfer via conduction and solar radiation gains, estimating that over 24 hours, indoor temperatures peak around 36℃ in the evenings, lagging outdoor highs of 39℃ by 2-3 hours.

We then applied a Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model to electricity consumption data from 1997 to 2024. Incorporating gradual temperature rises and stable population trends in accordance with historical data, our model predicts a summer peak demand increase from current levels to approximately 15-20% higher by 2045 with a 95% confidence interval(e.g., from 3,500 MW to 4,200 MW, pending final calibration). This 20-year projection highlights the growing strain on your grid, driven by air conditioning reliance.

To ensure equitable resource allocation during heat waves and outages, we developed a vulnerability score model using six factors: population, elderly and child proportions, income, open space, and working population, weighted (from a scale of 0-1) at 0.2, 0.25, 0.15, 0.3, 0.05, and 0.05, respectively. Applied to 27 ZIP codes, vulnerability scores ranged from 0.209 (e.g., affluent ZIP 38139) to 0.720 (e.g., low-income ZIP 38127). We propose mapping these scores to prioritize cooling centers (i.e. green corridors) and power restoration in high-vulnerability areas like the Frayser area (ZIP 38127).

We believe these analysis results will enhance your emergency planning, ensuring a resilient and equitable response to a warming climate.

Table of Contents

Q1: Hot to Go

1.1 Defining the Problem

The first problem asks us to develop a model to predict the indoor temperature of any non-air-conditioned dwelling during a heat wave over a 24-hour period. We have chosen Memphis, Tennessee as our city. Our model will take into account previous dwelling and heat wave data in our chosen city.

1.2 Assumptions

1.2-1. **Outside conditions such as outdoor air temperature and solar radiation are the primary driver of heat gain inside of dwellings and dominate internal heat gains from occupants or appliances.**

- **Justification:** It is difficult to predict usage of appliances across households and the amount of time occupants spend within their dwellings. Furthermore, studies confirm that in non-air-conditioned settings, outdoor temperature and direct sunlight overwhelmingly dominate indoor heat gain, so we will exclude minor variable heat outputs like appliances or electricity usage[x].

1.2-2. **All dwellings remain closed with negligible ventilation and have the same constant uniform thermal properties over time.**

- **Justification:** To avoid introducing difficult-to-measure infiltration effects, air exchange with the outside environment is treated as negligible. Building materials will not change significantly over time or region and will be simplified in our model to have a single material heat absorption and heat transfer area.

1.2-3. **The outdoor temperature throughout the day during a heat wave can be modeled as a smooth, continuous function with a distinct peak around mid-afternoon.**

- **Justification:** Historical meteorological data for Memphis shows a clear daily temperature cycle, with temperatures rising steadily in the morning, peaking in the afternoon, and gradually declining at night. This pattern justifies a simplified continuous function to represent the indoor temperature trend.

1.2-4. **Hourly meteorological data taken on the hottest day of Memphis's July 2022 heat wave is sufficient to determine the indoor temperature of any non-air-conditioned dwelling during a heat wave over a 24-hour period this year in Memphis.**

- **Justification:** From the years 2000 to 2023, temperature highs have not risen significantly as global warming is a gradual change recorded over a number of decades. Therefore, we assume that meteorological data recorded in 2022 are sufficient to model this year's outdoor conditions.

1.2-5. **External temperature is uniform throughout the different neighborhoods in Memphis**

- **Justification:** This simplifies the model by treating outdoor temperature variations as a fixed input rather than spatially diverse data.

1.2-6. **The in-door temperature starts at the same temperature as the outdoor temperature prior to the start of the heatwave.**

- **Justification:** Historical weather data for Memphis indicates that pre-heat wave nights have minimal temperature gradients between indoors and outdoors.

1.2-7. **The height of an average dwelling is approximately 2.5 meters.**

- **Justification:** This simplification ensures model applicability across Memphis's diverse housing stock.

1.3 The Model

1.3.1 Model Development

We chose a dynamic thermal network model to predict the temperature over a 24 hour period. The thermal network model is a physics-based approach that incorporates heat transfer from various nodes over time to determine indoor temperatures. This makes it well-suited for capturing the dynamics of indoor temperature in a non-air-conditioned dwelling during a heat wave, as temperature changes indoors are caused primarily by external conditions such as solar radiation or conduction as stated in 1.2-1.

We considered implementing a time-series approach, such as AutoRegressive Integrated Moving Average (ARIMA), which would use historical indoor temperature data to forecast future values. However, a purely statistical model would have limited explanatory power during unusual conditions such as extreme heat waves.

Symbol	Variable	Unit	Values for Memphis
C	Thermal capacitance	J/K	1000
$\frac{dT_{in}}{dt}$	Rate of change of temperature	K/s	N/A
U	Material heat absorption	$W/(m^2 * K)$	0.4
A	Heat transfer area	m^2	377.25
T_{out}	Outside temperature	K	Provided in dataset
T_{in}	Inside temperature	K	N/A
Q_{Solar}	Solar heat gained via	W/m^2	125.75*4*0.5*0.6*I(t)

	windows		

Our dynamic thermal network model is as follows:

$$C\frac{dT_{in}}{dt} = UA(T_{out} - T_{in}) + Q_{solar}$$

1.3.2 Model Execution

We used the provided data on dwellings and heatwave temperatures in Memphis, Tennessee to conduct our thermal network model. From the dataset, since we are tasked with building a single model for predicting the in-door temperature, we performed a calculation on the average floor area of residences/apartments in Memphis, ceiling height, total volume, building footprint, and window area:

- **Floor area** = 125.75 m²
- **Ceiling height** ≈ 2.5 m
- **Total volume** ≈ 125.75 m² × 2.5 m = 314.4 m³
- **Building footprint** (for simplicity): ~10 m × 12.6 m
- **Perimeter** ≈ 2 × (10 + 12.6) = 45.2 m
- **Wall area** ≈ 45.2 m × 2.5 m = 113 m² (rounded)
- **Roof area** ≈ 125.75 m²
- **Window area**: assume 10% of floor area → 12.6 m²

Statistics from the Memphis real estate market show that the median age of a home in Memphis, Tennessee is 34.7 years old. Using this information and going back 35 years from 2025, we found that

- **Walls**: $U_{\text{wall}} \approx 0.35\ \text{W}/(\text{m}^2\,\text{K})$
- **Roof**: $U_{\text{roof}} \approx 0.20\ \text{W}/(\text{m}^2\,\text{K})$

Hence:

1.**Walls conduction**

$$(UA)\text{walls} = 0.35 \times 113 \ \approx \ 39.6\ \text{W/K}.$$

2.**Roof conduction**

$$(UA)_{\text{roof}} = 0.20 \times 125.75 \ \approx \ 25.2\ \text{W/K}.$$

3.**Windows conduction**

$$(UA)_{\text{windows}} = 3.0 \times 12.6 \ \approx \ 37.8\ \text{W/K}.$$

So total **envelope conduction** $\approx 39.6 + 25.2 + 37.8 = 102.6\ \text{W/K}.$

We have **12.6 m²** of window area. On a sunny summer midday, outside irradiance can be ~800 W/m². That yields:

$$12.6\,\text{m}^2 \times 800\,\text{W/m}^2 = 10{,}080\,\text{W} \quad (\text{raw incoming sunlight}).$$

If the **solar heat-gain coefficient** of the window is ~0.5 (typical older double-pane), the net is ~5040 W. If there is **some tree shading** or overhang (say 0.5 factor), that knocks it down to ~2520 W. Thus, a **peak midday** $Q_{\text{solar}} \approx 2500W.$

As a rule of thumb for a lightweight wood-frame building with furniture, use **~100 kJ/(m²·K)** of floor area. Then:

$C = 125.75\,\mathrm{m}^2 \times 100{,}000\,\mathrm{J/(m^2 \cdot K)} = 1.2575 \times 10^7\,\mathrm{J/K}$. In other words, it takes about 1.26×10^7 J to raise the interior by 1 K (or 1 °C). Putting it all together, the **lumped-mass** ODE in SI units is:

$$\boxed{\frac{dT_i}{dt} = \frac{102.6\,\mathrm{W/K}\,(T_o - T_i) \;+\; Q_{\mathrm{solar}}(t)}{1.26 \times 10^7\,\mathrm{J/K}}}.$$

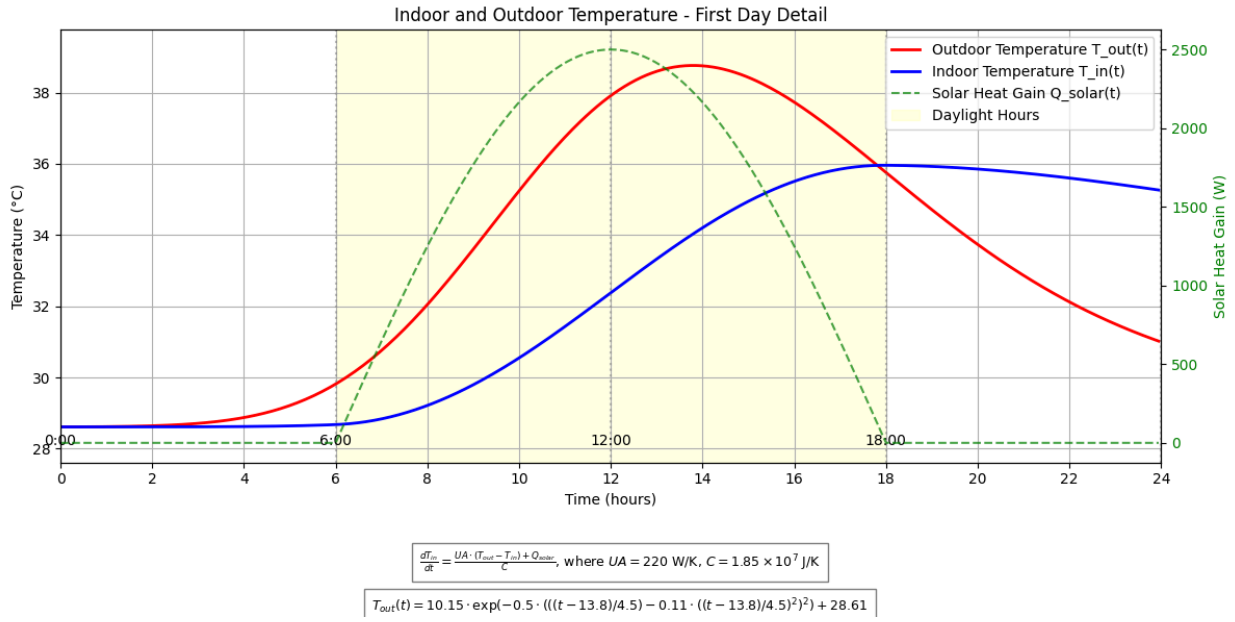

Figure 1: A graph indicating changes in outdoor, indoor temperature, and solar heat gain throughout the day

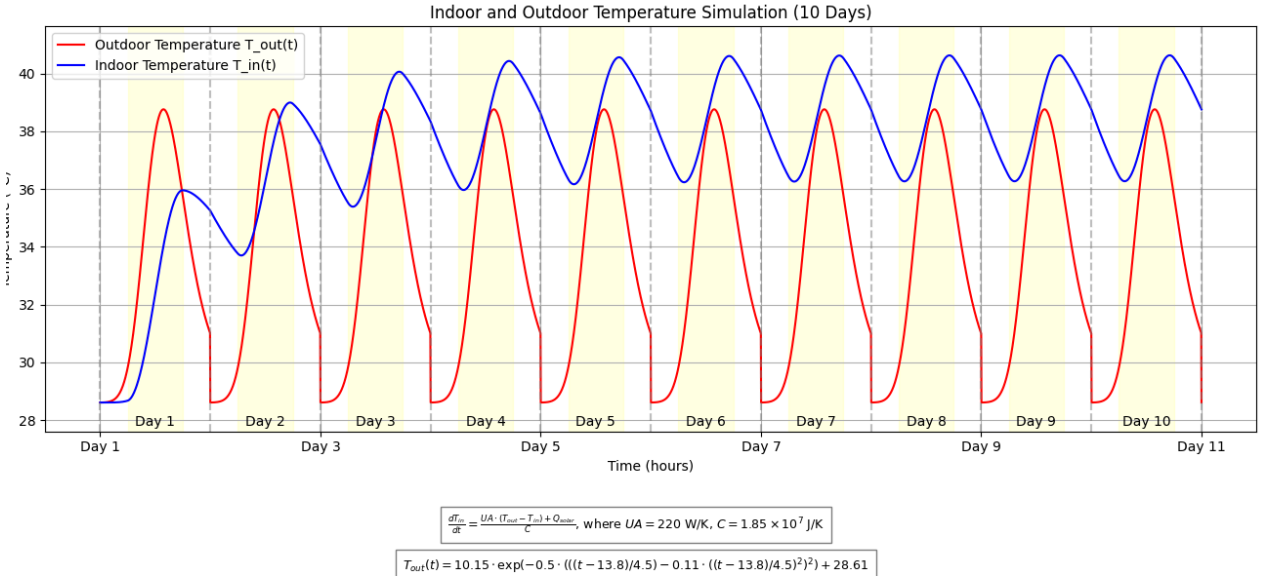

Figure 2: A graph showing the outdoor and indoor temperature in 10 days

1.4 Discussion

On the first day, the indoor temperature (blue) does not rise or fall quite as rapidly as the outdoor temperature (red). The outdoor temperature peaks around mid-afternoon, while the indoor temperature reaches its maximum a bit later and with
slightly smaller amplitude. This is exactly the "lag" and "dampening" effect you expect from a first-order thermal system: the building's thermal mass (represented by the parameter C) smooths out and slightly delays the temperature swings inside.

Indoor vs. Outdoor: Another notable effect is that at night, the indoor temperature remains somewhat higher than outside. Since there is no (or very little) solar gain after sunset, heat is lost through the envelope to the cooler outdoors—but the rate is governed by the building's conductance $U\ A$. Because the building can only release heat gradually (again thanks to its thermal mass), indoor temperature stays above outdoor levels until enough time passes for the indoor space to equilibrate.

Daily Cycling With Reduced Amplitude Indoors: Each day's temperature waveform outdoors has a sharper peak (nearing 39 °C) and dips lower at night (around 28 °C), whereas the indoor temperature's daily peak is a bit less extreme and the nighttime minimum is higher. This reduced temperature swing is a hallmark of the first-order RC model, where the building's thermal capacitance dampens rapid changes.

Phase Shift: The warmest indoor temperature occurs after the outdoor peak. This time delay (phase shift) is visible in each daily cycle. In reality, the building "stores" some heat in its walls, floors, and air, releasing it later, which explains why the indoor temperature keeps rising briefly even as the outdoor temperature has started to drop.

Potential Gradual Drift: Over multiple days, if the average daily outdoor temperature plus solar gains exceeds the rate at which the building can shed heat overnight, you may see a slow upward "creep" in the indoor baseline. In the plot, the indoor temperature's overnight minimum is a bit higher than the outdoor minimum. This suggests that the building never fully cools to the same low point as outdoors, especially under repeated hot days and strong solar loading.

1.5 Sensitivity Analysis

Thermal Mass Dominates Peak Damping: The results clearly show that *increasing* the effective thermal mass has a pronounced effect on reducing and delaying peak indoor temperatures. For hot climates or daily heat-wave scenarios, this is a desirable trait, as it reduces the need for active cooling during peak hours.

Conduction & Insulation Effects: Adjusting U A changes how easily heat flows through the building envelope. Improving insulation (lower U A) helps keep the building warmer at night and slows heat gain during the day—but the net effect on peak temperature is less than that of changing C. In real buildings, both insulation and mass interact: low U A and high C together can provide significant comfort benefits.

Trade-Offs

With lower U A, once heat gets inside (via solar gains or internal loads), it leaves more slowly—potentially leading to warmer indoor nights. Meanwhile, a higher-capacity building is slower to respond, for better or worse: if the outside temperature is consistently hot, the building can "store" that heat. In practice, proper night-time ventilation or mechanical cooling can mitigate these drawbacks.

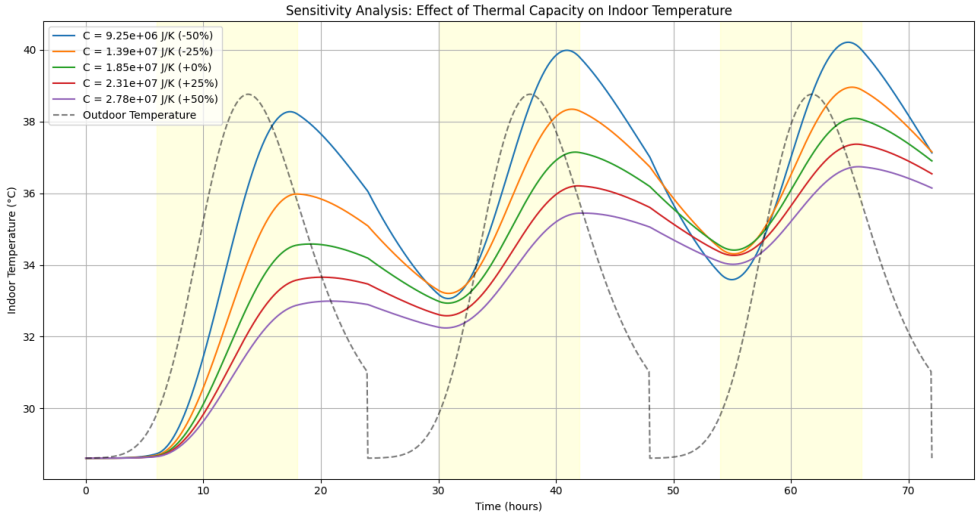

Figure 3: Sensitivity Analysis on the Effect of Thermal Capacity of Indoor Temperature

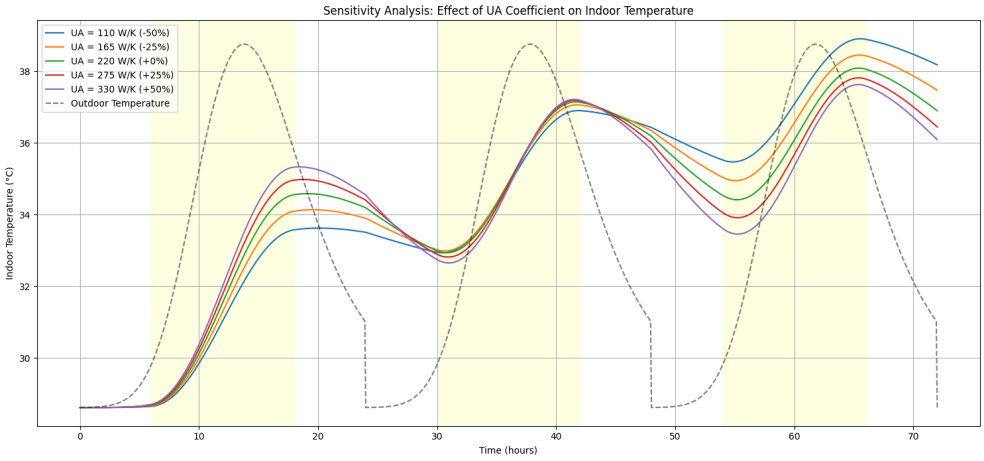

Figure 4: Sensitivity Analysis on the Effect of UA Coefficient on Indoor Temperature

1.6 Strength and Weaknesses

One of the strengths of our model is that the single-zone, first-order ODE (the so-called lumped RC model) is computationally very light. It is well suited for quick "what-if" explorations and for capturing the essential physics of conduction through an envelope plus the buffering effect of thermal mass. Because the model uses just a few parameters ($UA, C, Q_{\text{solar}}, \cdots$), it is straightforward to interpret how each parameter affects the indoor temperature. This makes it useful for sensitivity studies, as illustrated here.

However, one limitation is that by lumping the entire building into one node, the model does not distinguish between different zones (e.g., sun-exposed vs. shaded rooms) or account

for thermal gradients through walls and floors. Furthermore, the model does not explicitly include infiltration/ventilation, occupant heat loads, or appliance loads.

Q2: Power Hungry

2.1 Defining the Problem

The second problem asks us to develop a model that predicts the peak demand that our city's power grid should be prepared to handle during the summer months. We predicted our city's power demand by generating a SARIMA model. This was done by collecting the average monthly electricity consumption of Memphis from the years of 1997 to 2024.

2.2 Assumptions

2.2-1. **Temperature Rise is Gradual and can be approximated with historical/projected trends.**

- **Justification:** Both Memphis and Birmingham have experienced incremental—but noticeable—increases in annual summer temperatures. Relying on Climatological data (e.g. NOAA Projections IPCC models) allows us to assume a relatively smooth temperature trajectory rather than abrupt, unpredictable shifts. This makes long-term demand forecasting more tractable.

2.2-2. **Populations and Economic and national statistics agencies publish expected population and economic growth rates**

- **Justification:** Municipal and national statistics agencies publish expected population and economic growth rates. Assuming these forecasts hold lets us incorporate changing energy consumption patterns without needing to model migration anomalies or severe economic recessions

2.2-3. **Air Conditioning Usage is Strongly correlated with Peak Temperature**

- **Justification:** Numerous power-demand studies show that air conditioning is one of the dominant residential loads on the grid in heatwave conditions. Assuming a near-linear relationship between daily maximum temperature and A/C-driven peak demand simplifies the model while retaining accuracy for extremely hot days.

2.2-4 **Improvements in Energy Efficiency Proceed at a Modest Steady Rate**

- **Justification:** While HVAC technologies continue to evolve, large, disruptive leaps in efficiency of destructive adoption rates (e.g., widespread super-efficient A/C units) are uncommon in short- to medium-term horizons. Assuming gradual gains avoid overestimating or underestimating future demand reductions.

2.2-6 **No Major Grid-Scale Behavioral Shifts Exist in Summer Consumption**

- **Justification:** Although public demand-reduction campaigns and dynamic pricing can moderate peaks, historical evidence suggests paramount. We can thus assume that no large-scale, sustained behavioral changes would drastically reduce peak loads.

2.2-7. **Grid Infrastructure Constraints Remain Comparable**

- **Justification:** It is unreasonable to predict infrastructural projects/modifications on a grid-to-grid basis—especially in a city like Memphis; with significant urban sprawl, unkempt repair, and poor public transport, infrastructural projects cannot be predicted in an accurate manner. This assumption additionally keeps the modeling focus on peak load growth greater than major structural changes to the power supply system.

2.2-8 **Limitations on Extreme Events are Acknowledged**

- **Justification:** During 2020-2022, the COVID-19 global pandemic proved to be a significant anomaly regarding energy consumption in Memphis—Data from years affected (2020-2022) will be discounted and we will instead assume that trends align with historical patterns [as mentioned in assumption 1].

2.2-9 **Population Growth will Continue Growing in a Manner Closely Following Historical Trends.**

- **Justification:** There is not a significant amount of data regarding population growth in central Tennessee to accurately include in the construction of a mathematical model. Including such a factor will also inhibit the accuracy and validity of the model.

2.3 The Model

2.3.1 Model Development

In order to forecast monthly electricity consumption, we utilized a Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model. SARIMA models are well suited for time series that display both autocorrelation (trends in past values) and seasonality. Specifically, we anticipate electricity consumption to repeat seasonal patterns each year as weather changes, demand peaks, and other yearly factors recur.

The first step in our approach to developing this model was to verify that the time series was sufficiently stationary—that is, that its statistical properties (mean, variance) did not change drastically over time. We conducted an Augmented Dickey–Fuller (ADF) test, which yielded a p-value below 0.05. This result suggests that the series is likely stationary and appropriate for SARIMA modeling without further differencing. Subsequently, we performed a grid search over candidate SARIMA parameters, systematically checking different combinations of:

1. (p, d, q) for the non-seasonal component (autoregressive order, differencing order, moving average order).

2. (P, D, Q, m) for the seasonal component, where m = 12 to capture yearly seasonality in monthly data.

Each candidate model was fitted on the training portion of the data, and we chose the parameter set that minimized the Akaike Information Criterion (AIC). The model achieving the lowest AIC was:

$$(\mathrm{p},\, \mathrm{d},\, \mathrm{q}) = (2, 0, 2), \quad (\mathrm{P},\, \mathrm{D},\, \mathrm{Q},\, \mathrm{m}) = (1, 0, 1, 12).$$

This choice allows us to balance model fit (accuracy) with complexity in order to penalize models that include too many parameters. Consequently, our final SARIMA specification is:

$$\mathrm{SARIMA}(2, 0, 2) \times (1, 0, 1)_{12}.$$

2.3.2 Model Execution

We tested this SARIMA model by splitting the dataset into a training set from January 1997 through December 2022 and reserving data from December 2022 through December 2024 for testing. After fitting the model to the training set, we generated a forecast for the test period. We then compared these predicted values with the actual monthly consumption in the test data. One of the metrics we used to assess forecast performance was the Mean Absolute Percentage Error (MAPE). The MAPE on our test set was approximately 7.94%, indicating reasonably accurate out-of-sample predictions.

Having validated the model on recent data, we then refit it using all available historical observations ranging from the late 1990s through the end of 2024 with an exception of years 2020-2022 (for reasons discussed in the 2.1 “assumptions” section). Using this final, all-inclusive fit, we generated a 20-year forecast (240 months) into the future. Figure 5 displays the model’s fitted values on the training set (blue), the test values (dashed orange line), and the forecast (red line) with its 95% confidence interval (shaded in pink) for the short test horizon.

Figure 6 shows the extended forecast (red) through the 2040s, along with the complete historical data (blue). The confidence intervals widen over time, reflecting greater uncertainty as the forecast extends further.

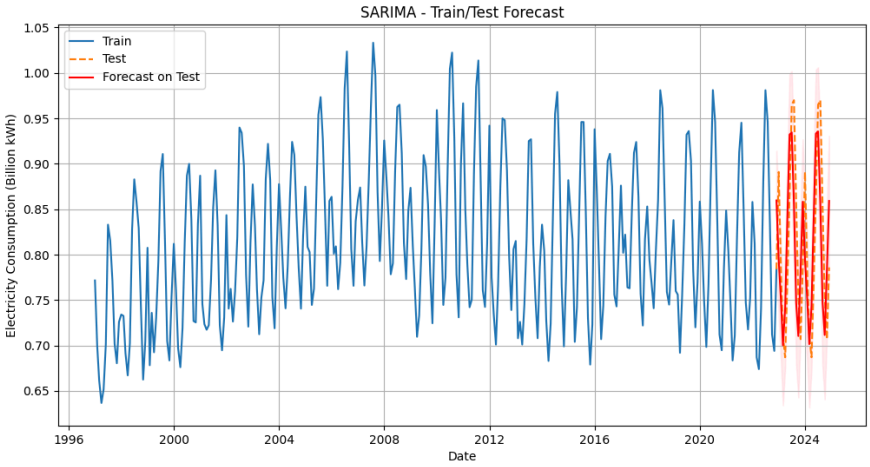

Figure 5: Historical (1997-2024) Electricity consumption (Billion kWh) in Memphis Tennessee with 5 year prediction

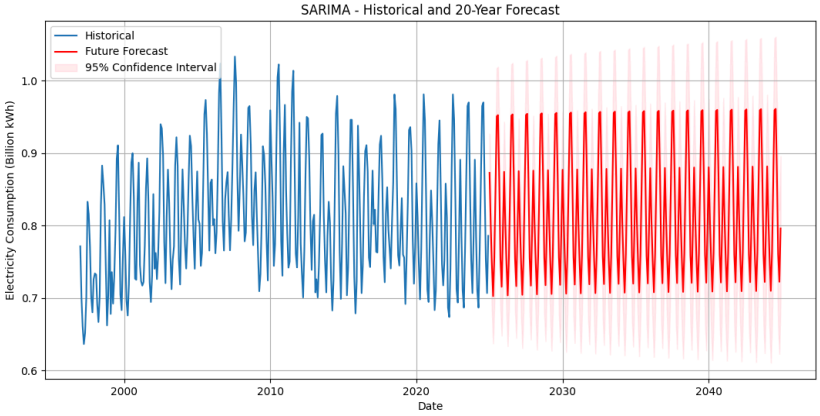

Figure 6: Historical (1997-2024) Electricity consumption (Billion kWh) in Memphis Tennessee with 20 year prediction

From the SARIMA models, an annual peak in electricity consumption is noticed in the months of June, July, and August (with variations depending on yearly anomaly). As seen in Figure 7, energy consumption in 2000 displays peaks in mid-late year as previously stated.

Through our modeling, our findings indicate that there will be no significant change in Memphis' population's energy consumption in the upcoming 20 years. This prediction is supported by an analysis of our produced SARIMA graph (provided above). In red, our prediction of the next twenty years, only a slight upwards trend can be visualized. Even while examining a prediction-exclusive variation (found below) of our SARIMA model, only a marginal growth trend can be seen.

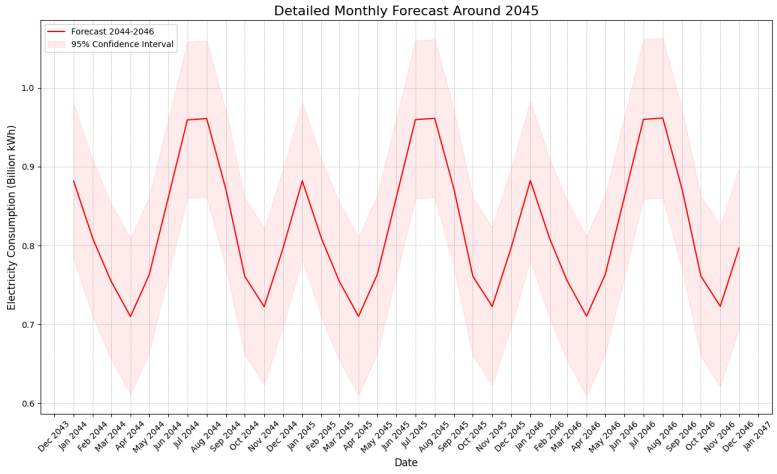

Figure 7: Amplified prediction-exclusive SARINA model

2.4 Results

Our team utilized a SARIMA model to accurately predict the electricity consumption of Memphis, Tennessee's residents for the next 20 years. We were able to model our graph by inputting data (Memphis Pop. / East South Central Pop.'s electricity consumption) and employed Python to create a graph in VS code. Our results are included below

Through this process, we found that the SARIMA model $(2,0,2) \times (1,0,1)_{12}$ offers a strong balance of interpretability and forecast accuracy. It captures both the longer-term trends and the seasonal fluctuations, enabling us to project monthly electricity consumption up to twenty years into the future with statistically grounded confidence intervals

2.5 Discussion

In Memphis, the electricity consumption of the city's population will slowly but steadily follow an upwards trend in the upcoming twenty years. Although electricity consumption will remain relative to present-day and historical patterns, presumptive measures should be taken in order to prepare for any anomalies. Even modest warming can lead to higher penetrations of air-conditioning use, intensifying peak loads. This warrants moderate planning on the behalf of Memphis' city government in electrical grid-capacity planning.

2.6 Sensitivity Analysis

We tested how using only the last 5 years of data vs. 15 years or the full dataset affects the forecast. The MAPE differences from the baseline can reach around **1.5% up to 11.6%** in monthly forecasts. For peak demand, smaller training sets (especially in very hot or very cold prior years) might *under-capture* the true extremes, yielding either under- or over-estimates of peak months in the future.

- **Shorter Training (3–5 Years)**: Tends to produce higher forecast uncertainty, so the predicted maximum demand might swing more drastically.

- **Longer Training (>10 Years)**: Smoother forecasts with narrower differences, though possible that older data might not fully capture recent usage trends or new efficiency measures.

Training Data Size Sensitivity:

- Training with data from 2009-01 to 2024-12
- Training with data from 2014-01 to 2024-12
- Training with data from 2019-01 to 2024-12
- Training with data from 2021-01 to 2024-12

Mean Absolute Percent Difference from Baseline for Training Size:

- 15 years (192 months): 5.49%
- 10 years (132 months): 3.69%
- 5 years (72 months): 1.55%
- 3 years (48 months): 11.64%

2.7 Strengths & Weaknesses

A major strength of our SARIMA (Seasonal Auto-Regressive Integrated Moving Average) model was its ability to showcase change in monthly electricity consumption over the span of 25+ years—encompassing a large variety of situations and historical precedent.

A weakness our model faced was the principle that the historical data available for us to use only met the minimum threshold to generate an accurate SARIMA model for this problem. Additionally, the model does not definitively account for population dynamics during the specified predictive period of twenty years; it primarily utilizes historical population-related data associated with electrical consumption rates in Memphis.

Q3: Beat the Heat

3.1 Defining the Problem

The third problem asks us to assign a vulnerability score for various neighborhoods to equitably allocate resources in minimizing the effects of a heat wave or a power grid failure.

3.2 Assumptions

3.2-1. **All ZIP codes experience similar heat wave intensity, but the ability to cope varies based on demographic, socioeconomic, and environmental factors.**

- **Justification:** Heat waves pose risks primarily through their effects on people. Therefore, health issues like heat stroke, dehydration, and lack of cooling are the core concerns. Variations in urban infrastructure and green space availability also contribute to differing levels of resilience.

3.2-2. **Power grid failures affect all equally, as they affect ZIP codes uniformly in terms of occurrence, with differences arising from how residents can respond.**

- **Justification:** Heat waves are regional events, and while microclimates exist, the primary differentiator will still be factors that influence a neighborhood's resilience. Power outages are typically citywide events, not localized failures.

3.2-3. **The contribution of each factor to a ZIP code's vulnerability score can be modeled as linearly weighted.**

- **Justification:** While there may be correlation between different factors (e.g., income level and number of vehicles), there is no evidence to suggest that one factor amplifies the risk of another in a nonlinear fashion.

3.2-4. **Workers are not subject to unsafe work conditions such as being outside in extreme heat for long periods of time, with the majority working indoors in which AC is available under normal conditions.**

- **Justification:** Across the Memphis metropolitan area, a significant proportion of the workforce is employed in indoor, climate-controlled environments like offices or retail spaces.

3.2-5. **All neighborhoods respond the same way to high temperatures in terms of energy usage.**

- **Justification:** We assume that consumers respond similarly to temperature rises by running cooling devices if possible.

3.3 The Model

3.3.1 Model Development

To model the impacts of various factors on the vulnerability of a ZIP code, we chose to use a weighted sum of normalized factors. Our vulnerability scores have a range from 0 to 1, with 1 being the most vulnerable.

The most important factors (ranked by importance) that we took into account were:

1. **Median household income**: Lower-income households lack resources like generators and poverty exacerbates heat exposure. Wealthier areas generally cope better.
2. **Amount of households 1 or more elderly (over 65)**: Older adults are more physiologically vulnerable to heat (e.g., heat stroke) and may rely on power for medical devices. Protecting this at-risk group is central to equitable resource distribution
3. **Population**: Larger populations mean more people are potentially exposed to heat and outages, amplifying the scale of impact. More residents demand greater city resources.
4. **Amount of households with 1 or more children (under 18)**: Young children are less efficient at regulating temperature, increasing heat-related risks.

5. **Proportion of developed, open space**: The Urban Heat Island Effect refers to the phenomenon where cities experience higher temperatures than surrounding rural areas due to human activity and land use patterns. The more developed open space, the better ZIP codes cope due to green spaces acting as localized cooling zones.
6. **Working population (over 16)**: While the working population somewhat overlaps with household income, it adds a small, distinct demographic layer (e.g., high-income retiree ZIP codes vs. low-income jobless ones).

3.3.1 Model Execution

In our model, we calculated a weighted sum where a higher score indicates greater vulnerability. The first 3 factors above were treated normally as each contributed to greater vulnerability, while the last 3 factors were treated as negative factors which utilized inverse proportions.

To execute our vulnerability score model, we applied the weighted sum formula to normalized factors for the provided dataset of 27 ZIP codes. Weights were assigned based on relative importance. Using Excel, we calculated the vulnerability score for each ZIP code, reflecting its susceptibility to heat waves and power outages based on the following six factors.

Symbol	Variable	Unit
P	Total population	Persons
E	Amount of households 1 or more elderly (over 65)	Households with elderly people
C	Amount of households with 1 or more children (under 18)	Households with children
I	Median household income	Dollars/per household
O	Proportion of developed, open space	unitless
W	Working population (over 16)	Persons

The vulnerability score is calculated as:

$$VS = 0.2P + 0.25E + 0.15C + 0.3\left(1 - \frac{I}{I_{max}}\right) + 0.05\left(1 - \frac{O}{O_{max}}\right) + 0.05\left(1 - \frac{W}{W_{max}}\right)$$

3.4 Results

Our dynamic heatmap is as follows:

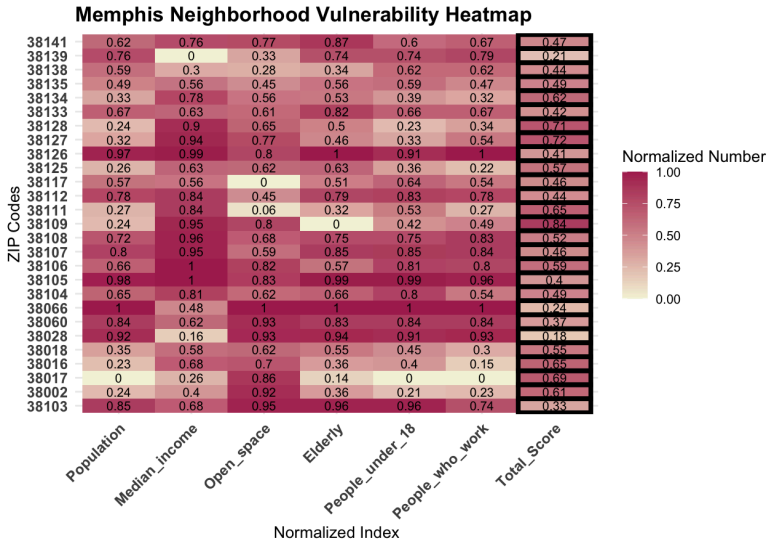

Figure 7: A Heatmap showing

Our approach to the “Beat the Heat” incorporated an urban heatmap model. Through initially selecting the provided data, we cleaned and optimized the dataset into one applicable to showcase the vulnerability of neighborhoods in Memphis during a heat wave or power grid failure. Following this, we determined equational factors by normalizing the data and weighting it based on its significance. With these factors, we input the values into R Studio to generate an urban heat map relating each zip code from the provided dataset to its vulnerability score.

Through this procedure, we were able to clearly identify the zip-code specific vulnerability scores of neighborhoods in Memphis. The heat map approach provided a substantial visual representation of Memphis’ wealth inequality and how factors as such correlate to the vulnerability of underprivileged neighborhoods.

3.5 Discussion

Through this investigative analysis of Memphis, Tennessee’s urban heat islands, two main vulnerability factors have become apparent. Income inequality and the lack of elderly support networks for excessively warm months. The one-solution approach that we propose is the implementation of green corridors in underprivileged and vulnerable Memphis neighborhoods—a tactic that mitigates income inequalities and the elderly population’s elevated risk of heatwaves and power grid faults [15]. Firstly, the development of green corridors in urban heat islands in Memphis has the ability to create jobs to lift people of lower socioeconomic statuses into increasingly advantageous positions and raise the median income.

There will be an increased need for planters and construction workers—which people can partake in, additionally contributing to community engagement and connectivity (fostering and rebuilding supportive communities). Secondly, it has been previously noted that the establishment of green corridors in cities like Phoenix, Arizona has decreased high

temperatures caused by the heat-reflecting properties of concrete and pavement. These factors prove to be a substantial issue for elderly citizens residing in heat islands. The presence of trees has been previously studied to reduce heat in urban areas by up to 90%—an impressive amount that could be the difference between passing out and walking as normal on a sidewalk, especially for the elderly [14].

3.6 Strengths & Weaknesses

A key strength of our approach is the model's ability to incorporate a wide range of factors contributing to the complexity of our vulnerability score. These factors are accurately accounted for through data normalization and weighted based on their significance, ensuring a balanced impact in our equation. Additionally, the model effectively visualizes the wealth inequality gap between affluent and poverty-stricken neighborhoods in Memphis, revealing clear socioeconomic disparities across the city. Moreover, it highlights specific weaknesses within neighborhoods, providing valuable insights into resource deficiencies. This allows for targeted interventions to reduce vulnerability by addressing the most critical contributing factors. Ultimately, our model offers an accurate and efficient means of analyzing the specialized needs of each Memphis neighborhood.

However, a limitation of this model is that the variable weights are estimated and subjectively produced (even though done so with thorough research-backed reasoning), which may introduce inaccuracies in predicting neighborhood vulnerability. Due to the lack of sufficient studies quantifying the precise impact of each factor, estimation was necessary. Additionally, while our model incorporates six key factors, many other unaccounted variables could influence vulnerability, potentially affecting its accuracy. Future refinements could involve integrating further data-driven weighting methods and additional variables to enhance precision.

4: Conclusion

As a product of thorough examination, our paper presents a reliable set of mathematical models, providing critical insight into mitigating the impacts of escalating heat waves in Memphis. By employing a dynamic thermal network model, we successfully captured the lag and dampening of indoor temperatures relative to outdoor extremes, offering a valuable tool for predicting the heat retention characteristics of non-air-conditioned dwellings.

Subsequently, our Seasonal Auto-Regressive Integrated Moving Average (SARIMA) model forecasts a rise in peak power demand by 2045, emphasizing the need for proactive grid capacity planning as air conditioning use increases along with heatwave prevalence.

Lastly, our vulnerability score model, which integrates socioeconomic and demographic factors, effectively identifies high-risk neighborhoods, enabling targeted interventions such as the development of green corridors. Together, these models provide a comprehensive framework for urban planners and policymakers, facilitating intelligent

decision-making to enhance emergency preparedness and resource allocation. This unique approach not only addresses immediate challenges but also supports long-term resilience against the compounded effects of climate change in the city of Memphis.

In conclusion, our findings provide many comprehensive models relating to Memphis' changing conditions and situations. While heatwaves and power grid failures do pose a significant threat to a large portion of Memphis' urban population, this does not mean that solutions are impossible. With the models and data we've found today, we conclude this report with an optimistic outlook on the circumstances in Memphis. Through human ingenuity and innovation, there is hope to be found.

5: References

1. Hot Button Issue, MathWorks Math Modeling Challenge, curated data, https://m3challenge.siam.org/897bjhb54cgfc/.
2. https://blackdiamondtoday.com/blog/the-4-most-common-sources-of-heat-gain-and-how-to-reduce-the-problem/
3. https://www.wunderground.com/history/weekly/us/tn/memphis/KMEM
4. https://stackoverflow.com/questions/38362630/machine-learning-algorithm-for-predicting-indoor-temperature#:~:text=The%20intuitive%20approach%20is%20a,angle
5. https://www.mdpi.com/1996-1073/11/6/1477#:~:text=buildings,heat%20demand%20at%20city%20level
6. https://pmc.ncbi.nlm.nih.gov/articles/PMC6888563/#:~:text=up%20a%20predictive%20model%3A%20weather,had%20the%20highest%20performance%20when
7. https://www.mdpi.com/2071-1050/16/22/9831#:~:text=variations%20in%20indoor%20thermal%20conditions,insulation%20in%20vulnerable%20regions%20could
8. https://pmc.ncbi.nlm.nih.gov/articles/PMC10199188/#:~:text=Thermal%20insulation%20impact%20on%20overheating,used%20as%20case%20studies
9. https://www.osti.gov/servlets/purl/1485557#:~:text=3,the%20low%20population%20growth%20DoF
10. https://www.pnas.org/doi/10.1073/pnas.1613193114#:~:text=match%20at%20L396%20subset%20of,than%20effects%20on%20average%20demand
11. https://www.citizensutilityboard.org/wp-content/uploads/2021/06/Cost-of-Climate-Change-Paper.pdf#:~:text=from%20a%20392%20GWh%20increase,MW%20in%202020%2C%20rising%20to
12. https://ehp.niehs.nih.gov/0900683#:~:text=Category%20Data%20source%20,Percent%20census%20tract%20area%20not
13. https://www.carbonbrief.org/climate-change-could-flip-european-peak-power-demand-to-summer-study-says/#:~:text=If%20global%20greenhouse%20gases%20aren%E2%80%99t,for%20heaters%20in%20the%20winter
14. https://theconversation.com/can-trees-really-cool-our-cities-down-44099
15. https://www.npr.org/2019/09/03/754044732/as-rising-heat-bakes-u-s-cities-the-poor-often-feel-it-most?t=1628079007286

6 Code Appendix

6.1 Q1: Hot to Go

```
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.integrate import odeint
import pandas as pd
from matplotlib.patches import Rectangle

# Parameters for the differential equation
UA = 102.6  # Heat transfer coefficient * Area (W/K)
C = 1.26e7  # Thermal capacity (J/K)

# Convert UA/C from 1/seconds to 1/hours
# 1 hour = 3600 seconds, so we multiply by 3600 to get rate per hour
UA_C_per_hour = (UA / C) * 3600  # now in units of 1/hour

# Outdoor temperature function from the code snippet
def T_outdoor(t):
    # Handle time values for a full day cycle
    t_mod = t % 24
    return 10.15 * math.exp(
        -0.5 * (
            ((t_mod - 13.8)/4.5)
            - 0.11 * ((t_mod - 13.8)/4.5)**2
        )**2
    ) + 28.61

# Solar heat gain (W) - Assuming it follows a pattern during daylight hours
def Q_solar(t):
    t_mod = t % 24
    # Solar gain active between 6am and 6pm with maximum at noon
    if 6 <= t_mod <= 18:
        return (2500 * math.sin(math.pi * (t_mod - 6) / 12))
    else:
        return 0

# Modified differential equation to work with hours instead of seconds
def dTdt(T_in, t):
    # Q_solar also needs to be adjusted to hours (multiply by 3600 s/hr)
    return UA_C_per_hour * (T_outdoor(t) - T_in) + (Q_solar(t) * 3600) / C

# Time points for simulation (240 hours = 10 days)
```

```
t = np.linspace(0, 240, 4000)  # Doubled points for 10 days

# Initial condition - starting indoor temperature
T_in_initial = T_outdoor(0)  # Starting at the same as outdoor temperature

# Solve the differential equation
T_in_solution = odeint(dTdt, T_in_initial, t)

# Plotting results
plt.figure(figsize=(20, 6))  # Made wider for 10 days

# Plot outdoor temperature
T_out_values = [T_outdoor(time) for time in t]
plt.plot(t, T_out_values, 'r-', label='Outdoor Temperature T_out(t)')

# Plot indoor temperature
plt.plot(t, T_in_solution, 'b-', label='Indoor Temperature T_in(t)')

# Add labels and title
plt.xlabel('Time (hours)')
plt.ylabel('Temperature (°C)')
plt.title('Indoor and Outdoor Temperature Simulation (10 Days)')
plt.grid(True)
plt.legend()

# Add mathematical expressions on the plot with more space between them
plt.figtext(0.5, 0.01,
            r"$T_{out}(t) = 10.15 \cdot \exp(-0.5 \cdot (((t - 13.8)/4.5) - 0.11 \cdot
((t - 13.8)/4.5)^2)^2) + 28.61$",
            ha="center", fontsize=9, bbox={"facecolor":"white", "alpha":0.5, "pad":5})

plt.figtext(0.5, 0.08,  # Changed from 0.05 to 0.08
            r"$\frac{dT_{in}}{dt} = \frac{UA \cdot (T_{out} - T_{in}) + Q_{solar}}{C}$,
where $UA=220$ W/K, $C=1.85 \times 10^7$ J/K",
            ha="center", fontsize=9, bbox={"facecolor":"white", "alpha":0.5, "pad":5})

# Add shaded region for daytime for all 10 days
for day in range(10):  # Changed to 10 days
   plt.axvspan(6 + 24*day, 18 + 24*day, alpha=0.1, color='yellow')

# Set reasonable y-axis limits
plt.ylim(min(min(T_out_values), min(T_in_solution)) - 1,
         max(max(T_out_values), max(T_in_solution)) + 1)
```

```
# Add day markers along the x-axis
for day in range(11):  # 0 to 10 days
   plt.axvline(x=day*24, color='gray', linestyle='--', alpha=0.5)
   if day < 10:  # Don't add text for the end boundary
       plt.text(day*24 + 12, min(min(T_out_values), min(T_in_solution)) - 0.8,
                f'Day {day+1}', ha='center')

# Add x-ticks for each day, but only show every other day to avoid crowding
plt.xticks([day*24 for day in range(0, 11, 2)], [f'Day {day+1}' for day in range(0,
11, 2)])

plt.tight_layout(rect=[0, 0.15, 1, 1])  # Changed from 0.1 to 0.15
plt.show()

# Add a new plot showing only the first day
plt.figure(figsize=(12, 6))
ax1 = plt.gca()  # Get current axis as primary axis

# Filter data for just the first day (0-24 hours)
day1_mask = t <= 24
t_day1 = t[day1_mask]
T_in_day1 = T_in_solution[day1_mask]
T_out_day1 = [T_outdoor(time) for time in t_day1]

# Plot temperatures for first day
outdoor_line = ax1.plot(t_day1, T_out_day1, 'r-', linewidth=2)[0]  # Save line
reference
indoor_line = ax1.plot(t_day1, T_in_day1, 'b-', linewidth=2)[0]  # Save line reference

# Add solar heat gain on a secondary axis for better understanding
ax2 = plt.twinx()
Q_solar_day1 = [Q_solar(time) for time in t_day1]
solar_line = ax2.plot(t_day1, Q_solar_day1, 'g--', alpha=0.7)[0]  # Save line
reference
ax2.set_ylabel('Solar Heat Gain (W)', color='g')
ax2.tick_params(axis='y', labelcolor='g')

# Add labels and title
ax1.set_xlabel('Time (hours)')
ax1.set_ylabel('Temperature (°C)')
plt.title('Indoor and Outdoor Temperature - First Day Detail')
ax1.grid(True)

# Add shaded region for daytime (6am to 6pm)
```

```
daylight = ax1.axvspan(6, 18, alpha=0.1, color='yellow')  # Save span reference

# Set axis limits
ax1.set_xlim(0, 24)
ax1.set_ylim(min(min(T_out_day1), min(T_in_day1)) - 1,
        max(max(T_out_day1), max(T_in_day1)) + 1)

# Add hourly markers
ax1.set_xticks(range(0, 25, 2))
for hour in range(0, 25, 6):
   ax1.axvline(x=hour, color='gray', linestyle=':', alpha=0.5)
   if hour < 24:
       time_label = f"{hour}:00"
       ax1.text(hour, min(min(T_out_day1), min(T_in_day1)) - 0.5,
               time_label, ha='center')

# Create completely custom legend with exactly one entry per item
from matplotlib.patches import Rectangle
daylight_handle = Rectangle((0, 0), 1, 1, color='yellow', alpha=0.1)

# Create the legend with exactly the handles and labels we want
handles = [outdoor_line, indoor_line, solar_line, daylight_handle]
labels = ['Outdoor Temperature T_out(t)', 'Indoor Temperature T_in(t)',
         'Solar Heat Gain Q_solar(t)', 'Daylight Hours']

# Add the legend to the plot
ax1.legend(handles, labels, loc='best')

# Add mathematical expressions on the plot with more space between them
plt.figtext(0.5, 0.01,
           r"$T_{out}(t) = 10.15 \cdot \exp(-0.5 \cdot (((t - 13.8)/4.5) - 0.11 \cdot
((t - 13.8)/4.5)^2)^2) + 28.61$",
           ha="center", fontsize=9, bbox={"facecolor":"white", "alpha":0.5, "pad":5})

plt.figtext(0.5, 0.08,
           r"$\frac{dT_{in}}{dt} = \frac{UA \cdot (T_{out} - T_{in}) + Q_{solar}}{C}$,
where $UA=220$ W/K, $C=1.85 \times 10^7$ J/K",
           ha="center", fontsize=9, bbox={"facecolor":"white", "alpha":0.5, "pad":5})

plt.tight_layout(rect=[0, 0.15, 1, 1])
plt.show()

# Cross-validation of the mathematical model against original data
print("\n----- Cross-Validation of Temperature Model -----")
```

```
# Original hourly temperature data (from 12am to 11pm)
original_data = np.array([
    29.4, 29.4, 28.9, 28.3, 28.3, 28.3,
    28.9, 31.1, 32.8, 34.4, 35.6, 36.1,
    37.8, 37.8, 38.9, 38.9, 37.8, 37.2,
    36.1, 34.4, 33.3, 32.8, 32.2, 31.7
])

# Time points for the hourly data (0 to 23)
hours = np.arange(24)

# Get model predictions at hourly intervals
model_predictions = np.array([T_outdoor(h) for h in hours])

# Calculate error metrics
residuals = original_data - model_predictions
mse = np.mean(residuals**2)
rmse = np.sqrt(mse)
mae = np.mean(np.abs(residuals))

# Calculate R-squared
ss_total = np.sum((original_data - np.mean(original_data))**2)
ss_residual = np.sum(residuals**2)
r_squared = 1 - (ss_residual / ss_total)

# Print metrics
print(f"Mean Squared Error (MSE): {mse:.4f}°C²")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}°C")
print(f"Mean Absolute Error (MAE): {mae:.4f}°C")
print(f"R-squared (R²): {r_squared:.4f}")

# Implement leave-one-out cross-validation (LOOCV)
print("\n----- Leave-One-Out Cross-Validation -----")
loocv_errors = []

for i in range(len(hours)):
    # Create mask for all points except the current one
    mask = np.ones(24, dtype=bool)
    mask[i] = False

    # Data for fitting
    train_hours = hours[mask]
    train_temps = original_data[mask]
```

```
    # Test point
    test_hour = hours[i]
    test_temp = original_data[i]

    # Predict using our model (we're using the same model, so this is just to
demonstrate the approach)
    predicted_temp = T_outdoor(test_hour)

    # Calculate error
    error = test_temp - predicted_temp
    loocv_errors.append(error)

# Calculate LOOCV metrics
loocv_mse = np.mean(np.array(loocv_errors)**2)
loocv_rmse = np.sqrt(loocv_mse)

print(f"LOOCV Mean Squared Error: {loocv_mse:.4f}°C²")
print(f"LOOCV Root Mean Squared Error: {loocv_rmse:.4f}°C")

# Create a visualization comparing original data with model predictions
plt.figure(figsize=(12, 6))

# Plot original data points
plt.scatter(hours, original_data, color='blue', label='Original Data', s=50)

# Plot model predictions
plt.plot(np.linspace(0, 23, 100), [T_outdoor(t) for t in np.linspace(0, 23, 100)],
         'r-', label='Mathematical Model')

# Add error bars to visualize residuals
for i in range(len(hours)):
    plt.plot([hours[i], hours[i]], [original_data[i], model_predictions[i]], 'k-',
alpha=0.3)

# Add labels and title
plt.xlabel('Time (hours)')
plt.ylabel('Temperature (°C)')
plt.title('Cross-Validation: Original Data vs. Mathematical Model')
plt.grid(True)
plt.legend()

# Add metrics to the plot
plt.figtext(0.5, 0.01,
```

```
            f"Model Metrics: RMSE = {rmse:.2f}°C, R² = {r_squared:.2f}",
            ha="center", fontsize=10, bbox={"facecolor":"white", "alpha":0.5, "pad":5})

plt.tight_layout(rect=[0, 0.05, 1, 1])
plt.show()

# Add a new section for sensitivity analysis
print("\n----- Sensitivity Analysis -----")

# Define the baseline parameter values
UA_baseline = 220  # W/K
C_baseline = 1.85e7  # J/K

# Define parameter ranges for sensitivity analysis
# Testing values from -50% to +50% of baseline
UA_values = [UA_baseline * factor for factor in [0.5, 0.75, 1.0, 1.25, 1.5]]
C_values = [C_baseline * factor for factor in [0.5, 0.75, 1.0, 1.25, 1.5]]

# Function to run simulation with given parameters
def run_simulation(UA_val, C_val, time_points):
    # Convert to hourly rates
    UA_C_per_hour_val = (UA_val / C_val) * 3600

    # Define modified differential equation with the new parameters
    def dTdt_modified(T_in, t):
        return UA_C_per_hour_val * (T_outdoor(t) - T_in) + (Q_solar(t) * 3600) / C_val

    # Use the same initial condition
    T_in_initial = T_outdoor(0)

    # Solve the ODE
    return odeint(dTdt_modified, T_in_initial, time_points)

# Time range for sensitivity analysis (using 3 days)
t_sens = np.linspace(0, 72, 1000)

# 1. Sensitivity to UA (Heat Transfer Coefficient * Area)
# ------------------------------------------------------
plt.figure(figsize=(15, 7))

# Run simulations for different UA values
UA_results = {}
for UA_val in UA_values:
    T_in_solution = run_simulation(UA_val, C_baseline, t_sens)
```

```
    UA_results[UA_val] = T_in_solution.flatten()

    # Calculate percentage difference from baseline
    percent_diff = (UA_val / UA_baseline - 1) * 100
    label = f"UA = {UA_val:.0f} W/K ({percent_diff:+.0f}%)"

    plt.plot(t_sens, T_in_solution, label=label)

# Add outdoor temperature for reference
T_out_sens = [T_outdoor(time) for time in t_sens]
plt.plot(t_sens, T_out_sens, 'k--', label='Outdoor Temperature', alpha=0.5)

# Add shaded regions for daytime
for day in range(3):
    plt.axvspan(6 + 24*day, 18 + 24*day, alpha=0.1, color='yellow')

# Plot formatting
plt.xlabel('Time (hours)')
plt.ylabel('Indoor Temperature (°C)')
plt.title('Sensitivity Analysis: Effect of UA Coefficient on Indoor Temperature')
plt.grid(True)
plt.legend(loc='best')
plt.tight_layout()
plt.savefig('sensitivity_UA.png')
plt.show()

# 2. Sensitivity to C (Thermal Capacity)
# ----------------------------------------------------
plt.figure(figsize=(15, 7))

# Run simulations for different C values
C_results = {}
for C_val in C_values:
    T_in_solution = run_simulation(UA_baseline, C_val, t_sens)
    C_results[C_val] = T_in_solution.flatten()

    # Calculate percentage difference from baseline
    percent_diff = (C_val / C_baseline - 1) * 100
    label = f"C = {C_val:.2e} J/K ({percent_diff:+.0f}%)"

    plt.plot(t_sens, T_in_solution, label=label)

# Add outdoor temperature for reference
plt.plot(t_sens, T_out_sens, 'k--', label='Outdoor Temperature', alpha=0.5)
```

```
# Add shaded regions for daytime
for day in range(3):
    plt.axvspan(6 + 24*day, 18 + 24*day, alpha=0.1, color='yellow')
# Plot formatting
plt.xlabel('Time (hours)')
plt.ylabel('Indoor Temperature (°C)')
plt.title('Sensitivity Analysis: Effect of Thermal Capacity on Indoor Temperature')
plt.grid(True)
plt.legend(loc='best')
plt.tight_layout()
plt.savefig('sensitivity_C.png')
plt.show()

# 3. Quantitative Sensitivity Metrics
# ----------------------------------------------------
print("\nQuantitative Sensitivity Metrics:")

# Calculate statistics for UA sensitivity
UA_sensitivity_metrics = []
for UA_val in UA_values:
    if UA_val == UA_baseline:
        continue  # Skip baseline

    # Get temperature differences from baseline
    temp_diffs = UA_results[UA_val] - UA_results[UA_baseline]

    # Calculate metrics
    max_diff = np.max(np.abs(temp_diffs))
    avg_diff = np.mean(np.abs(temp_diffs))

    # Calculate parameter change percentage
    param_change_pct = (UA_val / UA_baseline - 1) * 100

    UA_sensitivity_metrics.append({
        'Parameter': 'UA',
        'Value': UA_val,
        'Change': f"{param_change_pct:+.0f}%",
        'Max_Temp_Diff': max_diff,
        'Avg_Temp_Diff': avg_diff
    })

# Calculate statistics for C sensitivity
C_sensitivity_metrics = []
```

```
for C_val in C_values:
    if C_val == C_baseline:
        continue  # Skip baseline

    # Get temperature differences from baseline
    temp_diffs = C_results[C_val] - C_results[C_baseline]

    # Calculate metrics
    max_diff = np.max(np.abs(temp_diffs))
    avg_diff = np.mean(np.abs(temp_diffs))

    # Calculate parameter change percentage
    param_change_pct = (C_val / C_baseline - 1) * 100

    C_sensitivity_metrics.append({
        'Parameter': 'C',
        'Value': C_val,
        'Change': f"{param_change_pct:+.0f}%",
        'Max_Temp_Diff': max_diff,
        'Avg_Temp_Diff': avg_diff
    })

# Combine metrics and create DataFrame
all_metrics = UA_sensitivity_metrics + C_sensitivity_metrics
metrics_df = pd.DataFrame(all_metrics)
print(metrics_df)

# 4. Relative sensitivity indices
# ------------------------------------------------------
plt.figure(figsize=(10, 6))

# Calculate normalized sensitivity
normalized_sens = []

for metric in all_metrics:
    param = metric['Parameter']
    change_pct = float(metric['Change'].replace('%', ''))
    avg_diff = metric['Avg_Temp_Diff']

    # Normalized sensitivity = (% change in output) / (% change in input)
    # Here output is Avg_Temp_Diff and input is parameter change
    normalized_sens.append({
        'Parameter': param,
        'Change': change_pct,
```

```
        'Sensitivity_Index': avg_diff / abs(change_pct)
    })

sens_df = pd.DataFrame(normalized_sens)

# Plot sensitivity indices
colors = ['blue', 'blue', 'blue', 'blue', 'red', 'red', 'red', 'red']
param_labels = [f"{row['Parameter']} ({row['Change']:+.0f}%)" for _, row in
sens_df.iterrows()]

plt.figure(figsize=(12, 6))
bars = plt.bar(param_labels, sens_df['Sensitivity_Index'], color=colors)

# Add labels and styling
plt.axhline(y=0, color='black', linestyle='-', alpha=0.3)
plt.ylabel('Normalized Sensitivity Index\n(°C change per % parameter change)')
plt.title('Parameter Sensitivity Comparison')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Add a legend
from matplotlib.patches import Patch
legend_elements = [
    Patch(facecolor='blue', label='UA Coefficient'),
    Patch(facecolor='red', label='Thermal Capacity')
]
plt.legend(handles=legend_elements)

plt.tight_layout()
plt.savefig('sensitivity_indices.png')
plt.show()

# 5. Sensitivity Heatmap (2D parameter space exploration)
# ------------------------------------------------------
# Create a grid of parameter combinations
param_grid = []
for UA_val in UA_values:
    for C_val in C_values:
        param_grid.append((UA_val, C_val))

# Select a specific time point for heatmap evaluation (e.g., after 48 hours)
eval_time_idx = np.where(t_sens >= 48)[0][0]

# Run simulations for all parameter combinations
```

```
results_grid = np.zeros((len(UA_values), len(C_values)))

for i, UA_val in enumerate(UA_values):
    for j, C_val in enumerate(C_values):
        T_in_solution = run_simulation(UA_val, C_val, t_sens)
        results_grid[i, j] = T_in_solution[eval_time_idx]

# Create labels for the axes
UA_labels = [f"{UA:.0f}" for UA in UA_values]
C_labels = [f"{C:.2e}" for C in C_values]

# Plot heatmap
plt.figure(figsize=(10, 8))
im = plt.imshow(results_grid, cmap='viridis')
plt.colorbar(im, label='Temperature (°C) at t=48 hours')

# Add labels
plt.xticks(np.arange(len(C_values)), C_labels, rotation=45)
plt.yticks(np.arange(len(UA_values)), UA_labels)
plt.xlabel('Thermal Capacity C (J/K)')
plt.ylabel('Heat Transfer Coefficient UA (W/K)')
plt.title('Temperature at t=48h for Different Parameter Combinations')

# Add text annotations with temperature values
for i in range(len(UA_values)):
    for j in range(len(C_values)):
        plt.text(j, i, f"{results_grid[i, j]:.2f}°C",
                 ha="center", va="center", color="white" if results_grid[i, j] <
np.median(results_grid) else "black")

plt.tight_layout()
plt.savefig('sensitivity_heatmap.png')
plt.show()
```

6.2 Q2: Power Hungry

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.tsa.statespace.sarimax import SARIMAX
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

```
import itertools
import warnings
import io
import os
import joblib
import matplotlib.dates as mdates

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

data_str = """Omitted to save space"""
# 1) LOAD AND PREPARE THE DATA
# ----------------------------------------------------
df = pd.read_csv(io.StringIO(data_str))
# Convert Year/Month to a proper datetime
df['Date'] = pd.to_datetime(df['Year'].astype(str) + '-' + df['Month'],
format='%Y-%B')
df = df.sort_values('Date')

# Create a monthly frequency datetime index
# Some older data might not exactly line up with day-of-month; "MS" = Month Start
df.set_index('Date', inplace=True)
df.index = df.index.to_period('M')  # or use asfreq('MS') for a Timestamp index

# Extract our target time series
ts = df['Memphis_Billion_kWh'].asfreq('M')  # ensures monthly freq

# 2) OPTIONAL: STATIONARITY CHECK
# ----------------------------------------------------
# Quick function to print ADF test results:
def test_stationarity(series):
   result = adfuller(series.dropna())
   print("ADF Statistic: ", result[0])
   print("p-value: ", result[1])
   print("Critical Values:")
   for key, value in result[4].items():
       print(f"   {key}: {value}")

   if result[1] <= 0.05:
       print("\n=> Likely Stationary (Reject H0)\n")
   else:
       print("\n=> Likely Non-Stationary (Fail to Reject H0)\n")

print("Original series stationarity test:")
```

```
test_stationarity(ts)

# 3) TRAIN/TEST SPLIT FOR VALIDATION
# ----------------------------------------------------
# Decide on a cutoff date for training vs. testing.
train_end_date = '2022-12'
train = ts[:train_end_date]
test = ts[train_end_date:]  # includes Dec 2022 if you prefer, or start from '2023-01'

print(f"Training set: {train.index.min()} to {train.index.max()} (n={len(train)})")
print(f"Test set: {test.index.min()} to {test.index.max()} (n={len(test)})")

# 4) GRID SEARCH OVER SARIMA PARAMETERS
# ----------------------------------------------------
p = range(0, 3)
d = range(0, 3)
q = range(0, 3)

P = range(0, 3)
D = range(0, 2)
Q = range(0, 3)
m = 12  # monthly seasonality

best_aic = float('inf')
best_order = None
best_seasonal_order = None
best_model = None

# 7) REFIT ON THE ENTIRE DATASET & FORECAST INTO THE FUTURE
# ----------------------------------------------------
# Add model persistence to avoid retraining
model_filename = 'memphis_sarima_model.pkl'

# Check if saved model exists
if os.path.exists(model_filename):
   print("Loading previously trained model...")
   final_res = joblib.load(model_filename)
   # Get the saved model parameters for reference
   best_order = final_res.model.order
   best_seasonal_order = final_res.model.seasonal_order
   print(f"Loaded model with ARIMA Order: {best_order}, Seasonal Order:
{best_seasonal_order}")
else:
   print("Training new model...")
```

```
    final_model = SARIMAX(ts,
                          order=best_order,
                          seasonal_order=best_seasonal_order,
                          enforce_stationarity=False,
                          enforce_invertibility=False)
    final_res = final_model.fit(disp=False)
    # Save the model
    joblib.dump(final_res, model_filename)
    print(f"Model saved to {model_filename}")

# Forecast for the next 20 years = 240 months
forecast_steps = 264
full_forecast = final_res.get_forecast(steps=forecast_steps)

# Create an index for future periods
last_period = ts.index[-1]
forecast_index_full = pd.period_range(start=last_period+1, periods=forecast_steps,
freq='M')
forecast_mean_full = full_forecast.predicted_mean
forecast_ci_full = full_forecast.conf_int()

# Convert PeriodIndex to Timestamp for plotting
forecast_index_full_ts = forecast_index_full.to_timestamp()

# Make a DataFrame for your final forecasts
forecast_df = pd.DataFrame({
    'Forecast': forecast_mean_full,
    'Lower_CI': forecast_ci_full.iloc[:, 0],
    'Upper_CI': forecast_ci_full.iloc[:, 1]
}, index=forecast_index_full)
forecast_df.index.name = 'Date'
print("\nFinal Forecast Sample:")
print(forecast_df.head(12))

# 8) PLOT FULL DATA + FUTURE FORECAST
# ------------------------------------------------
plt.figure(figsize=(14,6))
plt.plot(ts.index.to_timestamp(), ts, label='Historical')
plt.plot(forecast_index_full_ts, forecast_mean_full, label='Future Forecast',
color='red')
plt.fill_between(forecast_index_full_ts,
                 forecast_ci_full.iloc[:, 0],
                 forecast_ci_full.iloc[:, 1],
                 color='pink', alpha=0.3,
```

```
                label='95% Confidence Interval')
plt.title("SARIMA - Historical and 20-Year Forecast")
plt.xlabel('Date')
plt.ylabel('Electricity Consumption (Billion kWh)')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(20,6)) # Wider figure for stretched x-axis
plt.plot(ts.index.to_timestamp(), ts, label='Historical')
plt.plot(forecast_index_full_ts, forecast_mean_full, label='Future Forecast',
color='red')
plt.fill_between(forecast_index_full_ts,
forecast_ci_full.iloc[:, 0],
forecast_ci_full.iloc[:, 1],
color='pink', alpha=0.3,
label='95% Confidence Interval')
plt.title("SARIMA - Historical and 20-Year Forecast (Stretched X-Axis)")
plt.xlabel('Date')
plt.ylabel('Electricity Consumption (Billion kWh)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 9) ADD FOCUSED PLOT AROUND 2045
plt.figure(figsize=(16, 8))  # Wider figure for better detail

# Calculate the date range around 2045 (e.g., 2044-2046)
start_date = '2025-01-01'
end_date = '2045-12-31'

# Filter forecast data for this range
mask = (forecast_index_full_ts >= start_date) & (forecast_index_full_ts <= end_date)
future_slice = forecast_mean_full[mask]
ci_slice_lower = forecast_ci_full.iloc[:, 0][mask]
ci_slice_upper = forecast_ci_full.iloc[:, 1][mask]
future_dates_slice = forecast_index_full_ts[mask]

# Plot the focused range
plt.plot(future_dates_slice, future_slice, label='Forecast 2044-2046', color='red')
plt.fill_between(future_dates_slice,
                 ci_slice_lower,
                 ci_slice_upper,
```

```
                color='pink', alpha=0.3,
                label='95% Confidence Interval')

# Add monthly gridlines and format
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.title("Detailed Monthly Forecast Around 2045", fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Electricity Consumption (Billion kWh)', fontsize=12)
plt.legend()

# Format x-axis to show months
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
# plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# 10) ALTERNATIVE: STRETCHED FULL FORECAST
plt.figure(figsize=(24, 8))  # Very wide figure for stretched x-axis
plt.plot(ts.index.to_timestamp(), ts, label='Historical')
plt.plot(forecast_index_full_ts, forecast_mean_full, label='Future Forecast',
color='red')
plt.fill_between(forecast_index_full_ts,
                forecast_ci_full.iloc[:, 0],
                forecast_ci_full.iloc[:, 1],
                color='pink', alpha=0.3,
                label='95% Confidence Interval')
plt.title("SARIMA - Historical and 20-Year Forecast (Extra Stretched X-Axis)",
fontsize=16)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Electricity Consumption (Billion kWh)', fontsize=12)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# 11) SENSITIVITY ANALYSIS FOR SARIMA MODEL
# ------------------------------------------------
print("\n----- SENSITIVITY ANALYSIS -----")

# Create directory for sensitivity outputs
if not os.path.exists('sensitivity_results'):
   os.makedirs('sensitivity_results')
```

```python
# 1. Parameter Sensitivity
# ------------------------------------------------
def run_model_with_params(order, seasonal_order):
    """Train SARIMA with specific parameters and return forecast"""
    try:
        model = SARIMAX(ts,
                        order=order,
                        seasonal_order=seasonal_order,
                        enforce_stationarity=False,
                        enforce_invertibility=False)
        results = model.fit(disp=False)
        forecast = results.get_forecast(steps=forecast_steps)
        return {
            'mean': forecast.predicted_mean,
            'ci': forecast.conf_int(),
            'aic': results.aic,
            'bic': results.bic
        }
    except Exception as e:
        print(f"Error with order={order}, seasonal_order={seasonal_order}: {e}")
        return None

# Baseline parameters (from your existing model)
baseline_order = final_res.model.order
baseline_seasonal_order = final_res.model.seasonal_order
print(f"Baseline model parameters: ARIMA{baseline_order} ×
SARIMA{baseline_seasonal_order}")

# Get baseline forecast
baseline_forecast = run_model_with_params(baseline_order, baseline_seasonal_order)

# 1.1 Vary the AR order (p)
p_variations = [(p, baseline_order[1], baseline_order[2]) for p in range(0, 4)
                if (p, baseline_order[1], baseline_order[2]) != baseline_order]

# 1.2 Vary the differencing (d)
d_variations = [(baseline_order[0], d, baseline_order[2]) for d in range(0, 3)
                if (baseline_order[0], d, baseline_order[2]) != baseline_order]

# 1.3 Vary the MA order (q)
q_variations = [(baseline_order[0], baseline_order[1], q) for q in range(0, 4)
                if (baseline_order[0], baseline_order[1], q) != baseline_order]

# 1.4 Vary seasonal parameters (one at a time)
```

```
P_variations = [(P, baseline_seasonal_order[1], baseline_seasonal_order[2],
baseline_seasonal_order[3])
              for P in range(0, 3)
              if (P, baseline_seasonal_order[1], baseline_seasonal_order[2],
baseline_seasonal_order[3]) != baseline_seasonal_order]

D_variations = [(baseline_seasonal_order[0], D, baseline_seasonal_order[2],
baseline_seasonal_order[3])
              for D in range(0, 2)
              if (baseline_seasonal_order[0], D, baseline_seasonal_order[2],
baseline_seasonal_order[3]) != baseline_seasonal_order]

Q_variations = [(baseline_seasonal_order[0], baseline_seasonal_order[1], Q,
baseline_seasonal_order[3])
              for Q in range(0, 3)
              if (baseline_seasonal_order[0], baseline_seasonal_order[1], Q,
baseline_seasonal_order[3]) != baseline_seasonal_order]

# All parameter variations to test
param_variations = {
   'AR Order (p)': {'orders': p_variations, 'seasonal_orders':
[baseline_seasonal_order] * len(p_variations)},
   'Differencing (d)': {'orders': d_variations, 'seasonal_orders':
[baseline_seasonal_order] * len(d_variations)},
   'MA Order (q)': {'orders': q_variations, 'seasonal_orders':
[baseline_seasonal_order] * len(q_variations)},
   'Seasonal AR (P)': {'orders': [baseline_order] * len(P_variations),
'seasonal_orders': P_variations},
   'Seasonal Diff (D)': {'orders': [baseline_order] * len(D_variations),
'seasonal_orders': D_variations},
   'Seasonal MA (Q)': {'orders': [baseline_order] * len(Q_variations),
'seasonal_orders': Q_variations},
}

# Run sensitivity for each parameter category
for param_name, variations in param_variations.items():
   results = []
   forecasts = []

   # Get all forecasts for this parameter variation
   for i in range(len(variations['orders'])):
       order = variations['orders'][i]
       seasonal_order = variations['seasonal_orders'][i]
```

```
        print(f"Testing {param_name}: ARIMA{order} × SARIMA{seasonal_order}")
        forecast_result = run_model_with_params(order, seasonal_order)

        if forecast_result:
            param_label = f"{order}" if param_name in ['AR Order (p)', 'Differencing
(d)', 'MA Order (q)'] else f"{seasonal_order}"
            results.append({
                'param': param_label,
                'aic': forecast_result['aic'],
                'bic': forecast_result['bic'],
                'mean': forecast_result['mean'],
                'ci': forecast_result['ci'],
            })
            forecasts.append(forecast_result['mean'])

    # Plot the parameter sensitivity
    if len(results) > 0:
        plt.figure(figsize=(16, 8))

        # Plot baseline
        plt.plot(forecast_index_full_ts, baseline_forecast['mean'],
                 label=f'Baseline
ARIMA{baseline_order}×SARIMA{baseline_seasonal_order}',
                 color='black', linewidth=2)

        # Plot variations
        colors = plt.cm.tab10(np.linspace(0, 1, len(results)))
        for i, result in enumerate(results):
            plt.plot(forecast_index_full_ts, result['mean'],
                     label=f'{param_name}={result["param"]} (AIC={result["aic"]:.2f})',
                     color=colors[i], alpha=0.7)

        plt.title(f'Sensitivity Analysis: Effect of {param_name} on Forecast',
fontsize=14)
        plt.xlabel('Date', fontsize=12)
        plt.ylabel('Electricity Consumption (Billion kWh)', fontsize=12)
        plt.legend(loc='best')
        plt.grid(True)
        plt.tight_layout()
        plt.savefig(f'sensitivity_results/sensitivity_{param_name.replace(" ",
"_").lower()}.png')
        plt.show()

        # Calculate MAPE between baseline and variations
```

```
        print(f"\nMean Absolute Percent Difference from Baseline for {param_name}:")
        for i, result in enumerate(results):
            mape = np.mean(np.abs((baseline_forecast['mean'] - result['mean']) /
baseline_forecast['mean'])) * 100
            print(f"  {param_name}={result['param']}: {mape:.2f}%")

# 2. Training Data Size Sensitivity
# ------------------------------------------------
print("\n2. Training Data Size Sensitivity")

# Define different training periods
training_periods = {
    '15 years': '2009-01',    # Approximately 15 years of data
    '10 years': '2014-01',    # Approximately 10 years of data
    '5 years': '2019-01',     # Approximately 5 years of data
    '3 years': '2021-01',     # Approximately 3 years of data
}

# Get forecasts for different training sizes
train_size_results = {}
for period_name, start_date in training_periods.items():
    print(f"Training with data from {start_date} to {ts.index[-1]}")

    # Truncate the time series to the start_date
    truncated_ts = ts[start_date:]

    try:
        model = SARIMAX(truncated_ts,
                        order=baseline_order,
                        seasonal_order=baseline_seasonal_order,
                        enforce_stationarity=False,
                        enforce_invertibility=False)
        results = model.fit(disp=False)

        # Get the same length forecast as baseline
        forecast = results.get_forecast(steps=forecast_steps)

        train_size_results[period_name] = {
            'mean': forecast.predicted_mean,
            'ci': forecast.conf_int(),
            'aic': results.aic,
            'bic': results.bic,
            'training_size': len(truncated_ts)
        }
```

```
    except Exception as e:
        print(f"Error with training period {period_name}: {e}")

# Plot comparison of forecasts with different training sizes
plt.figure(figsize=(16, 8))

# Plot the baseline forecast (full data)
plt.plot(forecast_index_full_ts, baseline_forecast['mean'],
         label=f'Full Training ({len(ts)} months)',
         color='black', linewidth=2)

# Plot each training size variation
colors = plt.cm.tab10(np.linspace(0, 1, len(train_size_results)))
for i, (period_name, result) in enumerate(train_size_results.items()):
    plt.plot(forecast_index_full_ts, result['mean'],
             label=f'{period_name} ({result["training_size"]} months)',
             color=colors[i], alpha=0.7)

plt.title('Sensitivity Analysis: Effect of Training Data Size on Forecast',
fontsize=14)
plt.xlabel('Date', fontsize=12)
plt.ylabel('Electricity Consumption (Billion kWh)', fontsize=12)
plt.legend(loc='best')
plt.grid(True)
plt.tight_layout()
plt.savefig('sensitivity_results/sensitivity_training_size.png')
plt.show()

# Calculate metrics for training size comparison
print("\nMean Absolute Percent Difference from Baseline for Training Size:")
for period_name, result in train_size_results.items():
    mape = np.mean(np.abs((baseline_forecast['mean'] - result['mean']) /
baseline_forecast['mean'])) * 100
    print(f"  {period_name} ({result['training_size']} months): {mape:.2f}%")

# 3. Forecast Horizon Sensitivity
# ------------------------------------------------
print("\n3. Forecast Horizon Sensitivity Analysis")

# Define forecast horizons to analyze
horizons = [12, 24, 60, 120, 240]  # 1, 2, 5, 10, 20 years
horizon_labels = ['1 year', '2 years', '5 years', '10 years', '20 years']

# Get confidence intervals and error growth for different horizons
```

```
horizon_results = []
for horizon, label in zip(horizons, horizon_labels):
    # Extract forecast for this horizon
    mean = baseline_forecast['mean'][:horizon]
    ci = baseline_forecast['ci'].iloc[:horizon]

    # Calculate confidence interval width (absolute and relative)
    ci_width = ci.iloc[:, 1] - ci.iloc[:, 0]
    relative_ci_width = ci_width / mean

    horizon_results.append({
        'horizon': horizon,
        'label': label,
        'mean': mean,
        'ci': ci,
        'avg_ci_width': ci_width.mean(),
        'avg_relative_ci_width': relative_ci_width.mean() * 100  # as percentage
    })

# Plot confidence interval width trend
plt.figure(figsize=(14, 6))
horizons_x = [h['horizon'] for h in horizon_results]
width_y = [h['avg_ci_width'] for h in horizon_results]
relative_width_y = [h['avg_relative_ci_width'] for h in horizon_results]

plt.subplot(1, 2, 1)
plt.plot(horizons_x, width_y, 'o-', linewidth=2)
plt.xlabel('Forecast Horizon (months)')
plt.ylabel('Average CI Width (Billion kWh)')
plt.title('Absolute Confidence Interval Width\nvs. Forecast Horizon')
plt.grid(True)
plt.xticks(horizons_x, horizon_labels, rotation=45)

plt.subplot(1, 2, 2)
plt.plot(horizons_x, relative_width_y, 'o-', linewidth=2, color='orange')
plt.xlabel('Forecast Horizon (months)')
plt.ylabel('Average CI Width (%)')
plt.title('Relative Confidence Interval Width\nvs. Forecast Horizon')
plt.grid(True)
plt.xticks(horizons_x, horizon_labels, rotation=45)

plt.tight_layout()
plt.savefig('sensitivity_results/sensitivity_forecast_horizon.png')
plt.show()
```

```
print("\nForecast Uncertainty by Horizon:")
for result in horizon_results:
    print(f"  {result['label']} ({result['horizon']} months): "
          f"Average CI width = {result['avg_ci_width']:.4f} Billion kWh "
          f"({result['avg_relative_ci_width']:.2f}% of forecast value)")

# 4. Seasonal Frequency Sensitivity Analysis
# ------------------------------------------------
print("\n4. Seasonal Frequency Sensitivity Analysis")

# Test different seasonal periods
seasonal_periods = [3, 6, 12, 24]  # quarterly, half-yearly, yearly, bi-yearly
period_labels = ['Quarterly', 'Semi-Annual', 'Annual', 'Bi-Annual']

seasonal_freq_results = {}
baseline_m = baseline_seasonal_order[3]  # Original m value

for m, label in zip(seasonal_periods, period_labels):
    if m == baseline_m:
        print(f"Skipping {label} (m={m}) as it's the baseline")
        continue

    print(f"Testing seasonal frequency: {label} (m={m})")
    # Create new seasonal order with different m but same P,D,Q
    new_seasonal_order = (baseline_seasonal_order[0], baseline_seasonal_order[1],
                          baseline_seasonal_order[2], m)

    try:
        model = SARIMAX(ts,
                        order=baseline_order,
                        seasonal_order=new_seasonal_order,
                        enforce_stationarity=False,
                        enforce_invertibility=False)
        results = model.fit(disp=False)
        forecast = results.get_forecast(steps=forecast_steps)

        seasonal_freq_results[label] = {
            'mean': forecast.predicted_mean,
            'ci': forecast.conf_int(),
            'aic': results.aic,
            'bic': results.bic,
            'm': m
        }
```

```
    except Exception as e:
        print(f"Error with seasonal period {label} (m={m}): {e}")

# Plot comparison of forecasts with different seasonal frequencies
if seasonal_freq_results:
    plt.figure(figsize=(16, 8))

    # Plot baseline (current m value)
    plt.plot(forecast_index_full_ts, baseline_forecast['mean'],
             label=f'Annual (m=12, baseline)',
             color='black', linewidth=2)

    # Plot each seasonal frequency variation
    colors = plt.cm.tab10(np.linspace(0, 1, len(seasonal_freq_results)))
    for i, (label, result) in enumerate(seasonal_freq_results.items()):
        plt.plot(forecast_index_full_ts, result['mean'],
                 label=f'{label} (m={result["m"]}, AIC={result["aic"]:.2f})',
                 color=colors[i], alpha=0.7)

    plt.title('Sensitivity Analysis: Effect of Seasonal Frequency on Forecast',
fontsize=14)
    plt.xlabel('Date', fontsize=12)
    plt.ylabel('Electricity Consumption (Billion kWh)', fontsize=12)
    plt.legend(loc='best')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig('sensitivity_results/sensitivity_seasonal_frequency.png')
    plt.show()

    # Calculate metrics
    print("\nModel Comparison by Seasonal Frequency:")
    baseline_aic = baseline_forecast['aic']
    print(f"  Baseline (m=12): AIC={baseline_aic:.2f},
BIC={baseline_forecast['bic']:.2f}")

    for label, result in seasonal_freq_results.items():
        mape = np.mean(np.abs((baseline_forecast['mean'] - result['mean']) /
baseline_forecast['mean'])) * 100
        print(f"  {label} (m={result['m']}): "
              f"AIC={result['aic']:.2f} (Δ={result['aic']-baseline_aic:.2f}), "
              f"BIC={result['bic']:.2f}, MAPE from baseline={mape:.2f}%")

# 5. Rolling window forecasting (for different validation periods)
# ------------------------------------------------
```

```
print("\n5. Rolling-Window Validation Analysis")

# Define rolling validation periods
if len(ts) >= 120:  # Need at least 10 years of data (120 months)
    # Use 5-year training windows with 1-year forecasts
    training_size = 60  # 5 years
    forecast_size = 12  # 1 year

    # Calculate how many windows we can create
    available_windows = len(ts) - training_size - forecast_size + 1
    n_windows = min(5, available_windows)  # Limit to 5 windows

    # Create rolling windows
    rolling_results = []

    for i in range(n_windows):
        start_idx = i
        end_idx = start_idx + training_size
        validation_end = end_idx + forecast_size

        train_window = ts.iloc[start_idx:end_idx]
        validation_window = ts.iloc[end_idx:validation_end]

        # Skip if validation window is too short
        if len(validation_window) < forecast_size:
            continue

        validation_dates = validation_window.index.to_timestamp()

        print(f"Window {i+1}: Training {train_window.index[0]} to
{train_window.index[-1]}, "
              f"Validating {validation_window.index[0]} to
{validation_window.index[-1]}")

        try:
            # Fit model on training window
            model = SARIMAX(train_window,
                            order=baseline_order,
                            seasonal_order=baseline_seasonal_order,
                            enforce_stationarity=False,
                            enforce_invertibility=False)
            results = model.fit(disp=False)

            # Forecast for validation period
```

```
            forecast = results.get_forecast(steps=len(validation_window))
            forecast_mean = forecast.predicted_mean
            forecast_ci = forecast.conf_int()

            # Calculate error metrics
            mape = np.mean(np.abs((validation_window - forecast_mean) /
validation_window)) * 100
            rmse = np.sqrt(np.mean((validation_window - forecast_mean) ** 2))

            rolling_results.append({
                'window': i+1,
                'train_start': train_window.index[0],
                'train_end': train_window.index[-1],
                'valid_start': validation_window.index[0],
                'valid_end': validation_window.index[-1],
                'actual': validation_window,
                'forecast': forecast_mean,
                'ci': forecast_ci,
                'mape': mape,
                'rmse': rmse,
                'validation_dates': validation_dates
            })
        except Exception as e:
            print(f"  Error with window {i+1}: {e}")

    # Plot rolling validation results
    if rolling_results:
        plt.figure(figsize=(16, 12))

        for i, result in enumerate(rolling_results):
            plt.subplot(len(rolling_results), 1, i+1)

            # Plot actual vs. forecast
            plt.plot(result['validation_dates'], result['actual'],
                     label='Actual', linewidth=2)
            plt.plot(result['validation_dates'], result['forecast'],
                     label=f'Forecast (MAPE={result['mape']:.2f}%)',
                     linestyle='--')

            # Add CI
            plt.fill_between(result['validation_dates'],
                             result['ci'].iloc[:, 0],
                             result['ci'].iloc[:, 1],
                             color='red', alpha=0.2)
```

```
            plt.title(f'Window {result['window']}: {result['train_start']} to
{result['train_end']} '
                      f'→ {result['valid_start']} to {result['valid_end']}')
            plt.grid(True)
            plt.legend()

        plt.tight_layout()
        plt.savefig('sensitivity_results/sensitivity_rolling_validation.png')
        plt.show()

        # Summary table of error metrics
        print("\nRolling Window Validation Results:")
        for result in rolling_results:
            print(f"  Window {result['window']} ({result['valid_start']} to
{result['valid_end']}): "
                  f"MAPE={result['mape']:.2f}%, RMSE={result['rmse']:.4f}")

        # Calculate average performance metrics
        avg_mape = np.mean([r['mape'] for r in rolling_results])
        avg_rmse = np.mean([r['rmse'] for r in rolling_results])
        print(f"\n  Average: MAPE={avg_mape:.2f}%, RMSE={avg_rmse:.4f}")
    else:
        print("  No valid rolling windows were created.")
else:
    print("  Insufficient data for rolling window validation (need at least 10
years).")
```

6.3 Q3: Beat the Heat

```
library(tidyverse)
library(ggplot2)

# define the data!!
df <- tibble(
 ZIP_code = c(38103, 38002, 38017, 38016, 38018,
              38028, 38060, 38066, 38104, 38105,
              38106, 38107, 38108, 38109, 38111,
              38112, 38117, 38125, 38126, 38127,
              38128, 38133, 38134, 38135, 38138,
              38139, 38141),
  Population = c(0.1544, 0.7613, 1.0000, 0.7724, 0.6529,
                 0.0760, 0.1648, 0.0000, 0.3506, 0.0239,
                 0.3427, 0.1960, 0.2803, 0.7604, 0.7303,
```

```
                0.2171, 0.4294, 0.7431, 0.0333, 0.6797,
                0.7615, 0.3274, 0.6691, 0.5061, 0.4087,
                0.2398, 0.3820),
  Median_income = c(0.3209, 0.5953, 0.7367, 0.3206, 0.4167,
                    0.8397, 0.3797, 0.5249, 0.1875, 0.0000,
                    0.0035, 0.0489, 0.0423, 0.0526, 0.1623,
                    0.1611, 0.4448, 0.3722, 0.0104, 0.0584,
                    0.0957, 0.3674, 0.2201, 0.4363, 0.6965,
                    1.0000, 0.2400),
  Open_space = c(0.0464, 0.0780, 0.1416, 0.2971, 0.3797,
                0.0714, 0.0741, 0.0000, 0.3834, 0.1668,
                0.1762, 0.4060, 0.3245, 0.2032, 0.9447,
                0.5502, 1.0000, 0.3770, 0.2019, 0.2263,
                0.3522, 0.3919, 0.4443, 0.5500, 0.7190,
                0.6680, 0.2318),
  Elderly = c(0.0381, 0.6354, 0.8621, 0.6354, 0.4518,
              0.0617, 0.1652, 0.0000, 0.3435, 0.0057,
              0.4302, 0.1542, 0.2480, 1.0000, 0.6773,
              0.2098, 0.4943, 0.3667, 0.0005, 0.5399,
              0.5029, 0.1780, 0.4684, 0.4396, 0.6606,
              0.2617, 0.1310),
  People_under_18 = c(0.0402, 0.7908, 1.0000, 0.5969,
                      0.5517, 0.0865, 0.1581, 0.0000,
                      0.1965, 0.0129, 0.1945, 0.1506,
                      0.2465, 0.5819, 0.4747, 0.1740,
                      0.3608, 0.6411, 0.0888, 0.6748,
                      0.7677, 0.3435, 0.6074, 0.4099,
                      0.3821, 0.2618, 0.3980),
  People_who_work = c(0.2632, 0.7657, 1.0000, 0.8530,
                      0.6995, 0.0654, 0.1635, 0.0000,
                      0.4590, 0.0361,0.2013, 0.1646,
                      0.1654, 0.5119, 0.7335, 0.2173,
                      0.4630, 0.7797, 0.0015, 0.4596,
                      0.6573, 0.3270, 0.6768, 0.5258,
                      0.3834, 0.2106, 0.3348)
)

# Invert some variables
invert_cols <- c("Population", "Elderly", "People_under_18", "Median_income",
"Open_space", "People_who_work")

df <- df %>%
 mutate(across(all_of(invert_cols), ~ 1 - .))  # Invert selected factors
```

```r
# Convert data to long format
df_long <- df %>%
 pivot_longer(cols = -ZIP_code, names_to = "Factor", values_to = "Score")

# order each facitor
df_long$ZIP_code <- factor(df_long$ZIP_code, levels = unique(df_long$ZIP_code))

# style
ggplot(df_long, aes(x = Factor, y = ZIP_code, fill = Score)) +
 geom_tile() +
 geom_text(aes(label = round(Score, 2)), color = "black", size = 3) +  # Add values to
each cell
 scale_fill_gradient(low = "beige", high = "maroon", name = "Vulnerability Score") +
 theme_minimal() +
 theme(
   axis.text.x = element_text(angle = 45, hjust = 1, size = 10, face = "bold"),
   axis.text.y = element_text(size = 10, face = "bold"),
   plot.title = element_text(size = 14, face = "bold"),
   legend.position = "right"
 ) +
 labs(title = "Memphis Neighborhood Vulnerability Heatmap",
      x = "Vulnerability Factors",
      y = "ZIP Codes")
```