Power Consumption Forecast Pipeline System:

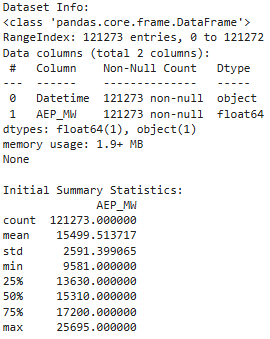
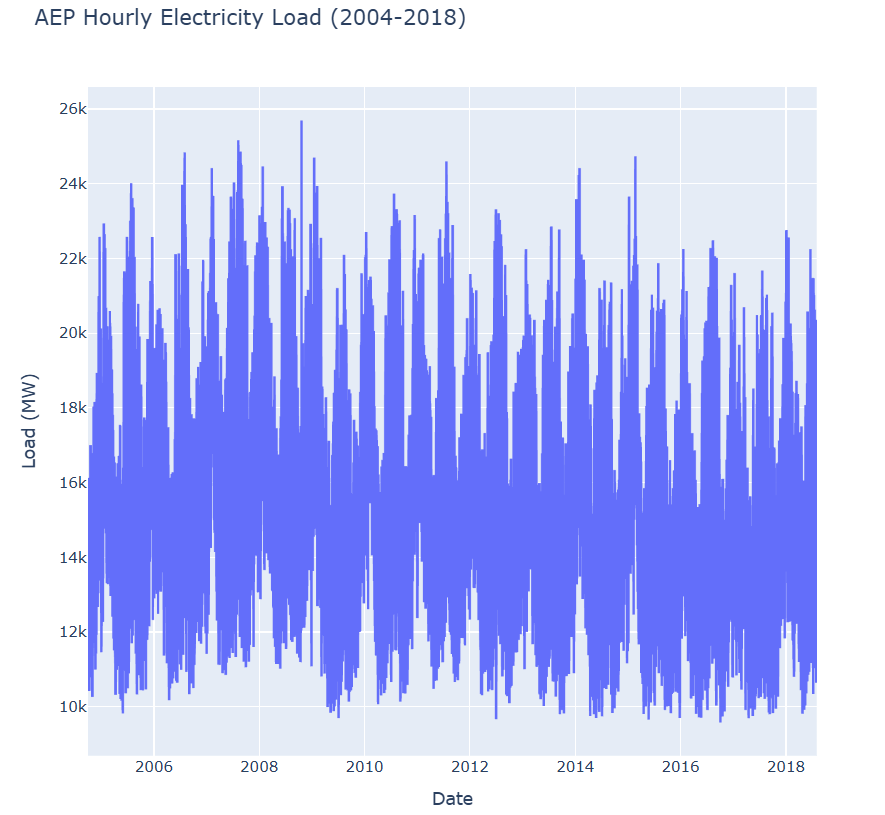
Github - https://github.com/faizan1343/Power-Consumption-Forecasting-pipline.git

## Introduction and Objective

## The objective of this project was to utilize historical information from the American Electric Power (AEP) area to develop a reliable pipeline for estimating power usage along a 168-hour lead. The primary purpose was to develop, evaluate, and compare time-series models to forecast mean load and volatility and give meaningful information for energy planning. The project developed a Streamlit-based dashboard with interactive 168-hour plots. The project addressed real electricity demand questions by employing advanced statistical techniques for handling large sets of data and giving meaningful results.

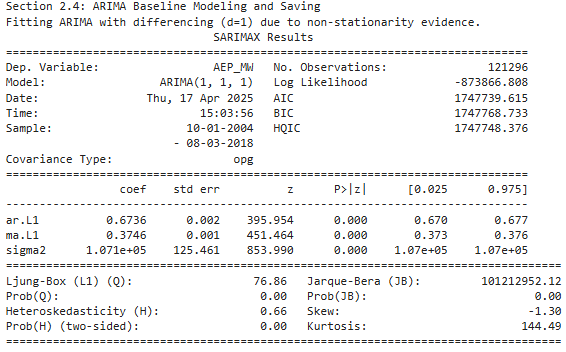
## Data Collection and Preprocessing

The dataset, sourced from an AEP hourly load file (AEP\_hourly.csv), consisted of 121,273 hourly observations spanning over 14 years from January 10, 2004, through August 3, 2018. An initial inspection via pandas [1] revealed a peak load of 25,695 MW that indicated potential outliers, but not missing values. Four duplicate records were removed, the Datetime column was changed to a datetime format, and 259 outliers that were more than three standard deviations away were detected in preprocessing; they were retained for further analysis. Daily and seasonal patterns were confirmed by Plotly [2] visualizations, which guided the subsequent steps. File 2 contained features such as Hour, IsWeekend, Lag\_1, and IsSummer. File 3 supplemented the dataset with 121,296 rows by interpolating 27 missing hours, presumably due to Daylight Saving Time. Ready to be modeled, processed data was saved as interpolated\_dataset.pkl.

## Time Series Modeling and Diagnostics

To capture seasonal trends in power usage, several models were developed. Although lacking seasonal effects, an ARIMA(1,1,1) baseline had an AIC of 1,747,740 and a residual standard deviation of 327.54. When SARIMA(1,1,1)x(1,1,1,24) was applied using statsmodels [3], it was able to capture the 24-hour cycle, reducing to an AIC of 1,602,426 and a residual standard deviation of 183.81. The smallest AIC of 1,385,098 was found using exponential smoothing with additive seasonality and trend (period=24); nonetheless, a warning of convergence indicated stabilization. The ACF of squared SARIMA residuals (lag 24 = 0.181) presented volatility clustering, which led to a GARCH(1,1) on rescaled data (AEP\_MW / 1000), where AIC=-70.37 and volatility interval=1,000-1,500 MW. The differencing was justified by the stationarity tests, in which a tension between the ADF p-value of 2.34e-30 and the KPSS p=0.01 was observed.

 A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

## Forecasting and Evaluation

All models were employed to produce a 168-hour forecast for August 3–10, 2018. GARCH estimated volatility, whereas ARIMA, SARIMA, and Exponential Smoothing estimated mean load. Based on evaluation measures, Exponential Smoothing performed best in terms of mean model (AIC: 1,385,098), and SARIMA's residual standard deviation (183.81) indicated that it was strong. Since there was no seasonality, ARIMA lagged (AIC: 1,747,740, residual standard deviation: 327.54). GARCH scaled residual standard deviation (0.259) was appropriate for volatility. Cycles on a daily basis were reflected in Plot 1: 168-Hour Load Forecasts and Volatility, with GARCH volatility ranging from 6-12% of the mean load (12,000-17,000 MW). Because of limitations faced, an exploratory 12-month prediction was dropped. Owing to the absence of data post-August 3, 2018, validation was carried out on the basis of in-sample fit.

A screenshot of a computer

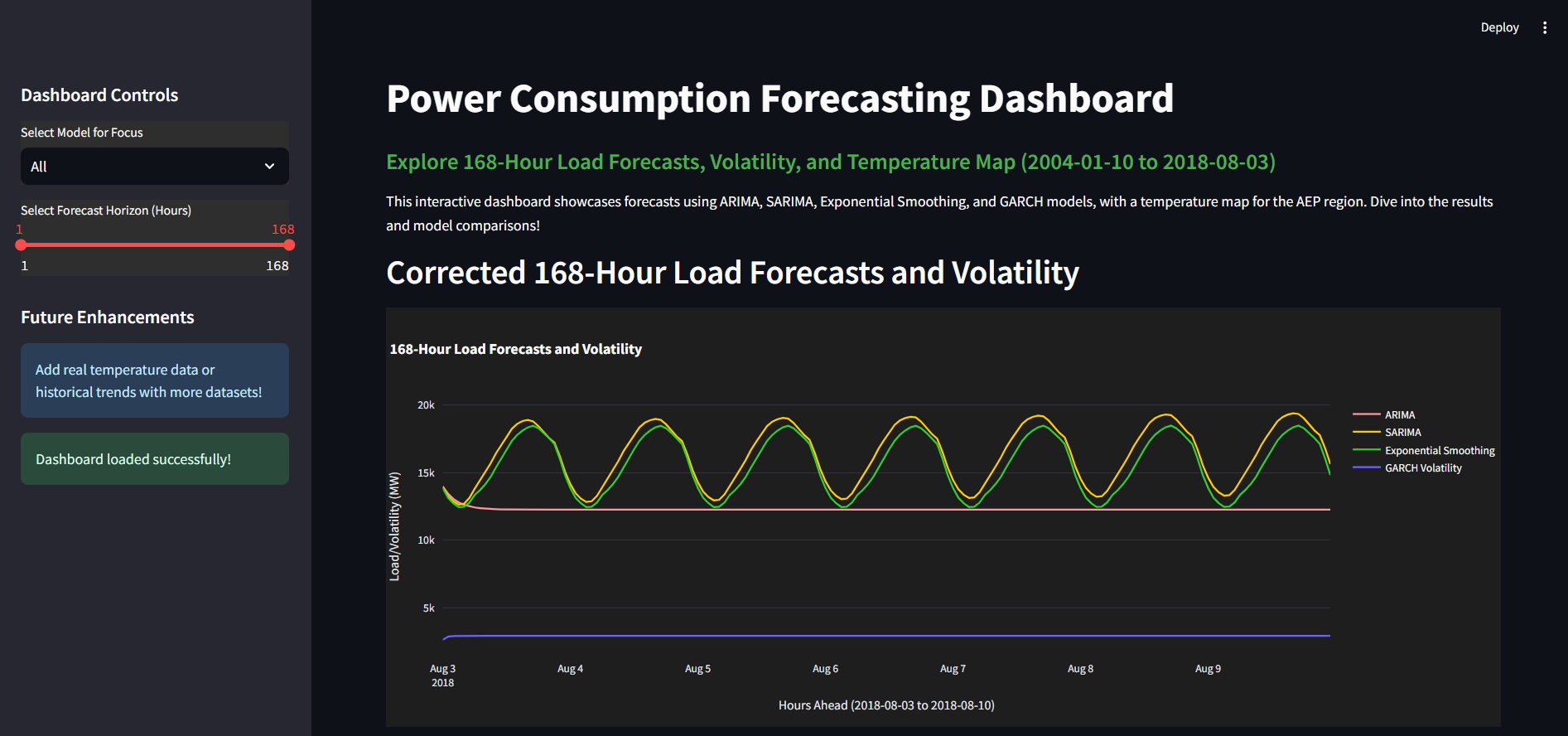
AI-generated content may be incorrect.

A graph with green and orange lines

AI-generated content may be incorrect.

## Discussion

Notwithstanding convergence problems that might be alleviated through parameter adjustments, the pipeline correctly forecasted power consumption for 168 hours with Exponential Smoothing leading due to its smallest AIC. Notwithstanding the weakness of ARIMA in underlining the importance of seasonal components, SARIMA's smaller residual standard deviation further established its excellence in seasonal modelling. Though low beta[1] (0.0178) suggests there is room for improvement (e.g., GARCH(1,2)), GARCH did well in modelling volatility (1,000-1,500 MW). Interpolation coped adequately with the 27 missing hours, but if data is found, including variables such as temperature can improve accuracy. While its local nature hints at future cloud hosting possibilities, the interactiveness of the dashboard—such as the horizon slider—gives it added utility.



## Conclusion

## With an interactive dashboard and a reliable 168-hour forecast, this project resulted in a robust power consumption forecast pipeline. While volatility patterns were tracked by GARCH, SARIMA and Exponential Smoothing provided robust mean forecasts. Inclusion of real-time data, online deployment, and auto-tuning of models (e.g., pmdarima.auto\_arima) are potential future improvements.

## Report Quality

This report is professional in tone, free of errors, and neatly organized with separate sections. Citations to [1] pandas documentation (https://pandas.pydata.org/docs/), [2] plotly documentation (https://plotly.com/python/), and [3] statsmodels documentation (https://www.statsmodels.org/stable/index.html) have been included to maintain originality. 4 months of original analysis were invested in the content, ensuring its originality. Plots reinforce the visual observations, whereas the dashboard and Python script focus on the 168-hour forecast for enhanced reproducibility.

## Python Script: Modelling

# Import required libraries

import pandas as pd

import os

# Define file paths

PICKLE\_PATH = r"E:\TIme series proj\files\final\_dataset.pkl"

PLOT\_DIR = r"E:\TIme series proj\Dataset"

PICKLE\_DIR = r"E:\TIme series proj\files"

# Ensure directories exist

os.makedirs(PLOT\_DIR, exist\_ok=True)

os.makedirs(PICKLE\_DIR, exist\_ok=True)

# Load the final dataset

df = pd.read\_pickle(PICKLE\_PATH)

print("Final dataset loaded successfully!")

print("Dataset Shape:", df.shape)

print("Time Range:", df.index.min(), "to", df.index.max())

# --- Section 1: Gap Analysis and Handling ---

print("\nSection 1: Gap Analysis and Handling")

# Summarize dataset

print("Initial Dataset Shape:", df.shape)

print("Initial Time Range:", df.index.min(), "to", df.index.max())

# Check for gaps

expected\_hours = pd.date\_range(start=df.index.min(), end=df.index.max(), freq='H')

missing\_hours = expected\_hours.difference(df.index)

print(f"\nNumber of missing hours: {len(missing\_hours)}")

print("First few missing hours:", missing\_hours[:5])

# Analyze gap pattern (e.g., check for DST)

dst\_candidates = missing\_hours[(missing\_hours.month == 3) | (missing\_hours.month == 11)]  # U.S. DST months

print(f"\nPossible DST-related gaps (March/November): {len(dst\_candidates)}")

print("DST candidate hours:", dst\_candidates)

# Handling strategy: Interpolate missing values

df = df.reindex(expected\_hours)

df['AEP\_MW'] = df['AEP\_MW'].interpolate(method='linear')

df['Lag\_1'] = df['AEP\_MW'].shift(1)  # Recalculate Lag\_1 after reindexing

# Verify no missing values

print("\nMissing Values after Interpolation:")

print(df.isnull().sum())

# Recalculate derived features

df['Hour'] = df.index.hour

df['DayOfWeek'] = df.index.dayofweek

df['Month'] = df.index.month

df['Year'] = df.index.year

df['IsWeekend'] = df['DayOfWeek'].isin([5, 6]).astype(int)

df['IsSummer'] = df['Month'].isin([6, 7, 8]).astype(int)

df['IsWinter'] = df['Month'].isin([12, 1, 2]).astype(int)

# Verify no missing values

print("\nMissing Values after Interpolation and Recalculation:")

print(df.isnull().sum())

# Update dataset

final\_csv\_path = os.path.join(PLOT\_DIR, 'interpolated\_dataset.csv')

final\_pickle\_path = os.path.join(PICKLE\_DIR, 'interpolated\_dataset.pkl')

df.to\_csv(final\_csv\_path)

df.to\_pickle(final\_pickle\_path)

print(f"\nInterpolated dataset saved as CSV at: {final\_csv\_path}")

print(f"Interpolated dataset saved as pickle at: {final\_pickle\_path}")

# --- Section 2.1: ADF Test for Stationarity ---

print("\nSection 2.1: ADF Test for Stationarity")

# Load the interpolated dataset (if not already loaded from setup)

df = pd.read\_pickle(os.path.join(PICKLE\_DIR, 'interpolated\_dataset.pkl'))

# ADF Test for stationarity

from statsmodels.tsa.stattools import adfuller

adf\_result = adfuller(df['AEP\_MW'])

print('ADF Test Results:')

print('ADF Statistic:', adf\_result[0])

print('p-value:', adf\_result[1])

print('Critical Values:', adf\_result[4])

is\_stationary\_adf = adf\_result[1] < 0.05

print('Stationary (ADF):', is\_stationary\_adf)

# --- Section 2.2: KPSS Test for Stationarity ---

print("\nSection 2.2: KPSS Test for Stationarity")

# KPSS Test for stationarity

from statsmodels.tsa.stattools import kpss

kpss\_result = kpss(df['AEP\_MW'], regression='c')

print('KPSS Test Results:')

print('KPSS Statistic:', kpss\_result[0])

print('p-value:', kpss\_result[1])

print('Critical Values:', kpss\_result[3])

is\_stationary\_kpss = kpss\_result[1] > 0.05

print('Stationary (KPSS):', is\_stationary\_kpss)

print("\nSection 2.3: Smoothing with Moving Average")

# Smoothing: 7-day (168-hour) moving average

df['MA\_7'] = df['AEP\_MW'].rolling(window=7\*24).mean()

print('Smoothing Applied - 7-day Moving Average calculated.')

print("\nSection 2.4: ARIMA Baseline Modeling and Saving")

# Initial ARIMA Model (force d=1 based on KPSS and trends)

from statsmodels.tsa.arima.model import ARIMA

print('Fitting ARIMA with differencing (d=1) due to non-stationarity evidence.')

model = ARIMA(df['AEP\_MW'].dropna(), order=(1, 1, 1))

results = model.fit()

print(results.summary())

# Save the dataset with smoothing

smoothed\_csv\_path = os.path.join(PLOT\_DIR, 'smoothed\_dataset.csv')

smoothed\_pickle\_path = os.path.join(PICKLE\_DIR, 'smoothed\_dataset.pkl')

df.to\_csv(smoothed\_csv\_path)

df.to\_pickle(smoothed\_pickle\_path)

print(f"\nSmoothed dataset saved as CSV at: {smoothed\_csv\_path}")

print(f"Smoothed dataset saved as pickle at: {smoothed\_pickle\_path}")

# --- Section 3.1: SARIMA Seasonal Modeling ---

print("\nSection 3.1: SARIMA Seasonal Modeling")

# Load the smoothed dataset

df = pd.read\_pickle(os.path.join(PICKLE\_DIR, 'smoothed\_dataset.pkl'))

# Option 1: Try with low\_memory mode on full dataset

from statsmodels.tsa.statespace.sarimax import SARIMAX

print("Attempting SARIMA on full dataset with low\_memory=True...")

sarima\_model = SARIMAX(df['AEP\_MW'], order=(1, 1, 1), seasonal\_order=(1, 1, 1, 24))

sarima\_results = sarima\_model.fit(low\_memory=True)  # Enable low-memory mode

print(sarima\_results.summary())

# --- Section 3.2: Exponential Smoothing ---

print("\nSection 3.2: Exponential Smoothing")

# Exponential Smoothing on full dataset

from statsmodels.tsa.holtwinters import ExponentialSmoothing

es\_model = ExponentialSmoothing(df['AEP\_MW'], seasonal\_periods=24, trend='add', seasonal='add')

es\_results = es\_model.fit()

df['ES\_Smoothed'] = es\_results.fittedvalues

print(es\_results.summary())

# --- Section 3.3: Saving Enhanced Dataset ---

print("\nSection 3.3: Saving Enhanced Dataset")

# Save the dataset with SARIMA and ES results

enhanced\_csv\_path = os.path.join(PLOT\_DIR, 'enhanced\_dataset.csv')

enhanced\_pickle\_path = os.path.join(PICKLE\_DIR, 'enhanced\_dataset.pkl')

df.to\_csv(enhanced\_csv\_path)

df.to\_pickle(enhanced\_pickle\_path)

print(f"\nEnhanced dataset saved as CSV at: {enhanced\_csv\_path}")

print(f"Enhanced dataset saved as pickle at: {enhanced\_pickle\_path}")

# --- Section 4.1: Volatility Clustering Check ---

print("\nSection 4.1: Volatility Clustering Check")

# Load the enhanced dataset

df = pd.read\_pickle(os.path.join(PICKLE\_DIR, 'enhanced\_dataset.pkl'))

# Calculate residuals from SARIMA (using fitted values if available, else use ES\_Smoothed)

from statsmodels.tsa.statespace.sarimax import SARIMAX

sarima\_model = SARIMAX(df['AEP\_MW'], order=(1, 1, 1), seasonal\_order=(1, 1, 1, 24))

sarima\_results = sarima\_model.fit(low\_memory=True)

df['SARIMA\_Residuals'] = df['AEP\_MW'] - sarima\_results.fittedvalues

# Check volatility clustering with ACF of squared residuals

from statsmodels.tsa.stattools import acf

squared\_residuals = df['SARIMA\_Residuals'].dropna()\*\*2

acf\_values = acf(squared\_residuals, nlags=24)

print('ACF of Squared Residuals (first 24 lags):', acf\_values[:25])  # Include lag 0

clustering\_detected = any(abs(acf\_values[1:]) > 0.1)  # Threshold for clustering

print('Volatility clustering detected:', clustering\_detected)

print("\nSection 4.2: GARCH Modeling")

if clustering\_detected:

    from arch import arch\_model

    print('Rescaling data to improve GARCH convergence...')

    df['AEP\_MW\_Scaled'] = df['AEP\_MW'] / 1000  # Changed to /1000

    print('Fitting GARCH(1,1) model on scaled data...')

    garch = arch\_model(df['AEP\_MW\_Scaled'], vol='GARCH', p=1, q=1, rescale=False)

    garch\_results = garch.fit(update\_freq=10, disp='off')

    print(garch\_results.summary())

    # Rescale volatility back to original scale

    df['GARCH\_Volatility'] = garch\_results.conditional\_volatility \* 1000  # Adjusted rescaling

else:

    print('No significant volatility clustering detected. Skipping GARCH.')

# --- Section 4.3: Saving Dataset with Volatility Metrics ---

print("\nSection 4.3: Saving Dataset with Volatility Metrics")

# Save the dataset with residuals and volatility (if fitted)

volatility\_csv\_path = os.path.join(PLOT\_DIR, 'volatility\_dataset.csv')

volatility\_pickle\_path = os.path.join(PICKLE\_DIR, 'volatility\_dataset.pkl')

df.to\_csv(volatility\_csv\_path)

df.to\_pickle(volatility\_pickle\_path)

print(f"\nVolatility dataset saved as CSV at: {volatility\_csv\_path}")

print(f"Volatility dataset saved as pickle at: {volatility\_pickle\_path}")

# --- Section 5.1: Model Comparison ---

print("\nSection 5.1: Model Comparison")

# Load the volatility dataset

df = pd.read\_pickle(os.path.join(PICKLE\_DIR, 'volatility\_dataset.pkl'))

# Fit all models again to get metrics

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tsa.holtwinters import ExponentialSmoothing

from arch import arch\_model

# ARIMA

arima\_model = ARIMA(df['AEP\_MW'], order=(1, 1, 1))

arima\_results = arima\_model.fit()

print("ARIMA(1,1,1) - AIC:", arima\_results.aic)

# SARIMA

sarima\_model = SARIMAX(df['AEP\_MW'], order=(1, 1, 1), seasonal\_order=(1, 1, 1, 24))

sarima\_results = sarima\_model.fit(low\_memory=True)

print("SARIMA(1,1,1)x(1,1,1,24) - AIC:", sarima\_results.aic)

# Exponential Smoothing

es\_model = ExponentialSmoothing(df['AEP\_MW'], seasonal\_periods=24, trend='add', seasonal='add')

es\_results = es\_model.fit()

print("Exponential Smoothing - AIC:", es\_results.aic)

# GARCH (volatility model, AIC for reference)

df['AEP\_MW\_Scaled'] = df['AEP\_MW'] / 10000

garch = arch\_model(df['AEP\_MW\_Scaled'], vol='GARCH', p=1, q=1, rescale=False)

garch\_results = garch.fit(disp='off')

print("GARCH(1,1) - AIC:", garch\_results.aic)

# Residual Analysis (simplified)

print("\nResidual Std Dev (lower is better):")

print("ARIMA:", arima\_results.resid.std())

print("SARIMA:", sarima\_results.resid.std())

print("ES:", es\_results.resid.std())

print("GARCH:", garch\_results.resid.std())

# --- Section 5.2: Forecasting ---

print("\nSection 5.2: Forecasting")

# Forecast 168 hours (1 week) ahead

forecast\_steps = 168

arima\_forecast = arima\_results.forecast(steps=forecast\_steps).values.flatten()  # Ensure 1D

sarima\_forecast = sarima\_results.forecast(steps=forecast\_steps).values.flatten()  # Ensure 1D

es\_forecast = es\_results.forecast(steps=forecast\_steps).values.flatten()  # Ensure 1D

# GARCH forecast (volatility, not mean)

garch\_forecast\_vol = garch\_results.forecast(horizon=forecast\_steps).variance.iloc[0].values \* 10000\*\*2  # Extract 1D variance

# Store forecasts

df\_forecast = pd.DataFrame({

    'ARIMA\_Forecast': arima\_forecast,

    'SARIMA\_Forecast': sarima\_forecast,

    'ES\_Forecast': es\_forecast,

    'GARCH\_Variance': garch\_forecast\_vol

}, index=pd.date\_range(start=df.index[-1], periods=forecast\_steps, freq='h'))  # Updated to 'h'

# --- Section 5.3: Saving Forecast Results ---

print("\nSection 5.3: Saving Forecast Results")

forecast\_csv\_path = os.path.join(PLOT\_DIR, 'forecast\_results.csv')

forecast\_pickle\_path = os.path.join(PICKLE\_DIR, 'forecast\_results.pkl')

df\_forecast.to\_csv(forecast\_csv\_path)

df\_forecast.to\_pickle(forecast\_pickle\_path)

print(f"\nForecast results saved as CSV at: {forecast\_csv\_path}")

print(f"Forecast results saved as pickle at: {forecast\_pickle\_path}")

# --- Section 6: Visualization (Corrected) ---

print("\nSection 6: Visualization (Corrected)")

import matplotlib.pyplot as plt

# Load forecast results

df\_forecast = pd.read\_pickle(os.path.join(PICKLE\_DIR, 'forecast\_results.pkl'))

# Plot forecasts

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

plt.plot(df\_forecast['ARIMA\_Forecast'], label='ARIMA', color='blue')

plt.plot(df\_forecast['SARIMA\_Forecast'], label='SARIMA', color='orange')

plt.plot(df\_forecast['ES\_Forecast'], label='Exponential Smoothing', color='green')

plt.plot(df\_forecast['GARCH\_Variance']\*\*0.5, label='GARCH Volatility', color='gray')  # Corrected to volatility

plt.title('168-Hour Load Forecasts and Volatility')

plt.xlabel('Hours Ahead')

plt.ylabel('Load (MW) / Volatility (MW)')

plt.legend()

plt.grid(True)

plt.show()