**Forecasting Hourly Electricity Consumption in the ComEd Region**

# Abstract:

This research aims to forecast hourly electricity consumption in the ComEd region by employing time series analysis methods. By understanding the consumption patterns, stakeholders can make informed decisions related to energy supply, grid management, and infrastructure planning.

# Background Information/Significance:

Electricity consumption patterns provide crucial insights into the behavior of consumers, the performance of the economy, and the effectiveness of energy policies. Accurate forecasting of these patterns is essential for ensuring a reliable energy supply, optimizing grid operations, and planning infrastructure investments. The ComEd region, serving the Chicago and Northern Illinois area, represents a significant portion of the state's population and economic activity, making the study of its consumption patterns of great importance.

# Initial Literature Review:

Several studies have highlighted the importance of electricity consumption forecasting for grid management and infrastructure planning. Time series analysis, especially methods like ARIMA and Exponential Smoothing, has been widely employed in this domain. Seasonal patterns, driven by factors like temperature, economic activity, and holidays, play a significant role in shaping electricity consumption.

# Hypothesis/Theories:

1. Hourly electricity consumption exhibits clear patterns that can be captured using time series analysis methods.
2. Seasonal components significantly influence electricity consumption, with potential peaks during specific times of the day or year.

# Research Methodology/Econometric Model:

* + **Data Decomposition**: Decompose the time series into trend, seasonality, and residual components.
  + **Model Selection**: Employ SARIMA and Exponential Smoothing to forecast electricity consumption.
  + **Validation**: Use a train-test split to validate the forecasting models' performance against actual observations.

# Data Sources:

The dataset is sourced from the "Hourly Energy Consumption" data, specifically focusing on the ComEd region. The data provides hourly electricity consumption readings spanning several years.

# Summary statistics and data cleaning procedure:

* + **Data Cleaning**: Check for missing values and handle them, if any.
  + **Summary Statistics**: Provide mean, median, standard deviation, and other statistics for hourly consumption.
  + **Visualization**: Plot time series graphs to visualize consumption patterns.

# Methodology:

# Dataset: We begin by importing the necessary Python libraries for data manipulation, visualization, and analysis. The dataset, 'COMED\_hourly.csv', contains historical hourly electricity consumption in megawatts (MW) for the ComEd region.

# A screenshot of a computer Description automatically generated

# Data Preprocessing: After loading the dataset into a pandas DataFrame, we examine the first few records to verify its structure. The data consists of two columns: 'Datetime' and 'COMED\_MW', with 'Datetime' indicating the date and time of the energy consumption record, and 'COMED\_MW' representing the electricity consumed during that hour.

# Initial Visualization: A preliminary plot of the entire dataset is created to visualize the electricity consumption trend over time. This helps in understanding the overall pattern and detecting any outliers or anomalies in the data.

# A graph showing a blue line Description automatically generated

# Indexing Time Series: We then set the 'Datetime' column as the index of the DataFrame and convert it to a pandas DateTime object to facilitate time series analysis.

# Outlier Detection: Checking for missing values reveals no gaps in the data. A histogram plot of 'COMED\_MW' helps in identifying and visualizing outliers. Records where 'COMED\_MW' is less than 19,000 MW are plotted separately to emphasize these outliers.

# A graph showing a blue line Description automatically generated

# Outlier Treatment: We remove these outliers from the dataset to prevent them from skewing the analysis.

# Feature Engineering: A function named create\_features is defined to extract time-related features from the DateTime index. These features include the hour, day of the week, quarter, month, year, day of the year, day of the month, and week of the year. Additionally, we derive a 'season' feature to represent the time of the year, categorized into Winter, Spring, Summer, and Fall.

# A screenshot of a calendar Description automatically generated

# Exploratory Data Analysis:

# A graph showing the energy consumption by hour Description automatically generated

# A graph of different colored squares Description automatically generated

# A graph showing the energy consumption by year Description automatically generated

# A graph showing different colored squares Description automatically generated

# A graph showing the energy consumption Description automatically generated

# Our data has seasonality.

# Daily peak/highest is around 6 PM and the mininum is at 4 AM

# Least energy consumption on weekends (Saturday/Sunday)

# The highest energy consumption in a year is either in the end of the year or in the middle of the year

# No significant trend or change in total energy consumption throughout the year 2002-2018

# Highest energy consumption in summer, then winter

# Data Simplification: To focus on the univariate aspect of the time series, we create a simplified version of the dataset. This involves removing all features except for 'COMED\_MW', resulting in a dataset that captures electricity consumption over time without additional temporal attributes.

# A white background with red text Description automatically generated

# Daily Aggregation: The hourly records are resampled to a daily frequency, calculating the average electricity consumption for each day. This step reduces the granularity of the data, which can help in identifying broader trends and smoothing out short-term fluctuations.

# Visualization of Daily Data: We visualize the daily resampled time series, which provides a clearer picture of the long-term trends and seasonal patterns in electricity consumption.

# A graph showing time and date time Description automatically generated

# Train-Test Split and Plotting them: The dataset is split into training and test sets to validate the forecasting model's performance. The training set includes data up to the end of 2014, while the test set comprises data from 2015 onwards. A plot is created to illustrate the division between the training and test data, with a clear demarcation at the start of the year 2015. This visualization confirms that we have a substantial amount of data for both model training and evaluation.

# The ARIMA (AutoRegressive Integrated Moving Average) modeling approach is a cornerstone of time series forecasting and is used extensively for univariate time series analysis where data points are serially correlated.

# Step 1 — Check Stationarity: The first step in ARIMA modeling is to ensure that the time series is stationary. Stationarity means the statistical properties of the series do not change over time. This is important because ARIMA models require the data to be stationary to make forecasts.

# Step 2 — Differencing: If the series is not stationary, it needs to be transformed to achieve stationarity. This process is called differencing. We take the first difference of the series, then test for stationarity again. Seasonal differencing is also considered if seasonality is present.

# Step 3 — Validation Sample: We set aside a portion of the data as a validation sample. This subset is used to test the model's accuracy and will not be used in the training phase.

# Step 4 — Model Selection: The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots are used to identify the appropriate AR and MA terms for the ARIMA model.

# Step 5 — Model Construction: We construct the ARIMA model using the identified parameters and train it on the dataset, setting the number of forecast periods as required.

# Step 6 — Model Validation: The model's predictive accuracy is assessed by comparing the forecasted values against the actual values in the validation sample.

# Using the seasonal\_decompose function, we observed high seasonality in the data, indicating that a seasonal ARIMA (SARIMA) model might be more appropriate. The test\_stationarity function, which utilizes the Dickey-Fuller test, confirmed that the time series is stationary, allowing us to proceed without further differencing.

# The ARIMA model's parameters were determined using the auto\_arima function from the pmdarima library, which identified an SARIMAX(1, 0, 1)x(2, 1, 1, 7) as the best model. Despite the SARIMAX model's ability to account for seasonality and integrate terms, the resulting forecasts from the model did not perform as well as expected, with a reported RMSE of 4681.8977. The comparison of actual and predicted values showed discrepancies, suggesting that further model refinement or a different modeling approach might be necessary to handle the multiple seasonalities in the data.

# We then check the seasonality and observed that “we have very high seasonality which is not ideal”

# A graph of blue lines Description automatically generated with medium confidence

# We then checked if the time series is stationary:

# A graph showing a number of data Description automatically generated with medium confidence

# We use a rolling mean and standard deviation of 24 hours and plot these along with the original time series. If the rolling statistics do not change over time, it is an indication that the time series is stationary. We also perform the Dickey-Fuller test and check if the p-value is less than 0.05. If it is, we can reject the null hypothesis that the time series is non-stationary.

# Based on the results of the Dickey-Fuller test, the p-value is less than 0.05, and we can reject the null hypothesis. Therefore, the time series is stationary.

# ACF and PCF Plots:

# A graph of a graph showing a number of blue lines Description automatically generated with medium confidence

# Model Building, Fitting, Testing (Predictions) and Evaluation:

# Step 1 — Model Optimization with auto\_arima:

# Using the auto\_arima function, we conducted a stepwise search to identify the best SARIMAX model based on the Akaike Information Criterion (AIC). Starting from the simplest combination of parameters, the function iteratively increased the complexity of the model.

# Step 2 — Seasonal Decomposition:

# A seasonal decomposition was performed on the training dataset with a period of 90 days, highlighting a strong seasonal pattern.

# Step 3 — Stationarity Check:

# Rolling statistics and the Dickey-Fuller test were used to verify the stationarity of the time series. The rolling mean and standard deviation plots alongside the original series indicated stationarity, confirmed by a p-value much less than 0.05 from the Dickey-Fuller test.

# Step 4 — ACF and PACF Analysis:

# The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots were generated to guide the selection of AR and MA terms for the ARIMA model.

# Step 5 — Fitting the SARIMAX Model:

# A SARIMAX model was fitted to the training data with the order (1, 0, 1) and seasonal order (2, 1, 1, 7), chosen based on the previous analyses. The L-BFGS-B optimization algorithm was used for parameter estimation.

# Model Summary:

# The summary of the SARIMAX model showed significant coefficients for the autoregressive term (ar.L1), moving average term (ma.L1), and the seasonal moving average term (ma.S.L7). A relatively high sigma-squared value indicates variability in the model residuals.

* Performing stepwise search to minimize aic
* ARIMA(0,0,0)(1,1,1)[7] intercept : AIC=90712.791, Time=3.40 sec
* ARIMA(0,0,0)(0,1,0)[7] intercept : AIC=91549.953, Time=0.29 sec
* ARIMA(1,0,0)(1,1,0)[7] intercept : AIC=86213.508, Time=6.96 sec
* ARIMA(0,0,1)(0,1,1)[7] intercept : AIC=86938.647, Time=5.63 sec
* ARIMA(0,0,0)(0,1,0)[7] : AIC=91547.963, Time=0.16 sec
* ARIMA(1,0,0)(0,1,0)[7] intercept : AIC=87510.621, Time=0.72 sec
* ARIMA(1,0,0)(2,1,0)[7] intercept : AIC=85700.698, Time=13.59 sec
* ARIMA(1,0,0)(2,1,1)[7] intercept : AIC=85104.787, Time=31.90 sec
* ARIMA(1,0,0)(1,1,1)[7] intercept : AIC=inf, Time=18.22 sec
* ARIMA(1,0,0)(2,1,2)[7] intercept : AIC=inf, Time=34.34 sec
* ARIMA(1,0,0)(1,1,2)[7] intercept : AIC=inf, Time=21.45 sec
* ARIMA(0,0,0)(2,1,1)[7] intercept : AIC=90705.185, Time=8.49 sec
* ARIMA(2,0,0)(2,1,1)[7] intercept : AIC=inf, Time=34.58 sec
* ARIMA(1,0,1)(2,1,1)[7] intercept : AIC=84496.276, Time=31.97 sec
* ARIMA(1,0,1)(1,1,1)[7] intercept : AIC=inf, Time=16.91 sec
* ARIMA(1,0,1)(2,1,0)[7] intercept : AIC=85170.052, Time=15.37 sec
* ARIMA(1,0,1)(2,1,2)[7] intercept : AIC=inf, Time=42.51 sec
* ARIMA(1,0,1)(1,1,0)[7] intercept : AIC=85629.281, Time=8.44 sec
* ARIMA(1,0,1)(1,1,2)[7] intercept : AIC=inf, Time=37.08 sec
* ARIMA(0,0,1)(2,1,1)[7] intercept : AIC=86921.097, Time=17.52 sec
* ARIMA(2,0,1)(2,1,1)[7] intercept : AIC=inf, Time=38.22 sec
* ARIMA(1,0,2)(2,1,1)[7] intercept : AIC=inf, Time=36.05 sec
* ARIMA(0,0,2)(2,1,1)[7] intercept : AIC=85603.880, Time=20.40 sec
* ARIMA(2,0,2)(2,1,1)[7] intercept : AIC=inf, Time=44.57 sec
* ARIMA(1,0,1)(2,1,1)[7] : AIC=inf, Time=24.51 sec
* Best model: ARIMA(1,0,1)(2,1,1)[7] intercept
* Total fit time: 513.374 seconds
* SARIMAX Results
* =========================================================================================
* Dep. Variable: y No. Observations: 4748
* Model: SARIMAX(1, 0, 1)x(2, 1, 1, 7) Log Likelihood -42241.138
* Date: Mon, 11 Dec 2023 AIC 84496.276
* Time: 17:41:46 BIC 84541.524
* Sample: 01-01-2002 HQIC 84512.178
* - 12-31-2014
* Covariance Type: opg
* ==============================================================================
* coef std err z P>|z| [0.025 0.975]
* ------------------------------------------------------------------------------
* intercept -24.5373 4.826 -5.085 0.000 -33.996 -15.079
* ar.L1 0.8240 0.007 112.958 0.000 0.810 0.838
* ma.L1 0.3930 0.012 32.897 0.000 0.370 0.416
* ar.S.L7 0.0012 0.014 0.087 0.931 -0.026 0.029
* ar.S.L14 0.0164 0.014 1.177 0.239 -0.011 0.044
* ma.S.L7 -0.9098 0.008 -107.967 0.000 -0.926 -0.893
* sigma2 3.315e+06 5.42e+04 61.196 0.000 3.21e+06 3.42e+06
* ===================================================================================
* Ljung-Box (L1) (Q): 6.64 Jarque-Bera (JB): 1070.16
* Prob(Q): 0.01 Prob(JB): 0.00
* Heteroskedasticity (H): 1.03 Skew: 0.13
* Prob(H) (two-sided): 0.55 Kurtosis: 5.31
* ===================================================================================

# Convergence and Results:

# The optimization converged successfully, as indicated by the message "CONVERGENCE: REL\_REDUCTION\_OF\_F\_<=\_FACTR\*EPSMCH". The AIC of 83936.735 suggests the model's relative quality, with lower AIC values representing a better fit to the historical data.

* RUNNING THE L-BFGS-B CODE
* \* \* \*
* Machine precision = 2.220D-16
* N = 6 M = 10
* At X0 0 variables are exactly at the bounds
* At iterate 0 f= 8.96759D+00 |proj g|= 1.04244D-01
* At iterate 5 f= 8.92980D+00 |proj g|= 6.26931D-02
* At iterate 10 f= 8.91131D+00 |proj g|= 3.45786D-02
* At iterate 15 f= 8.90934D+00 |proj g|= 1.76991D-03
* At iterate 20 f= 8.90934D+00 |proj g|= 5.72570D-04
* At iterate 25 f= 8.90925D+00 |proj g|= 9.86515D-03
* At iterate 30 f= 8.90153D+00 |proj g|= 1.15422D-01
* At iterate 35 f= 8.83813D+00 |proj g|= 4.38090D-02
* At iterate 40 f= 8.83790D+00 |proj g|= 2.31707D-03
* \* \* \*
* Tit = total number of iterations
* Tnf = total number of function evaluations
* Tnint = total number of segments explored during Cauchy searches
* Skip = number of BFGS updates skipped
* Nact = number of active bounds at final generalized Cauchy point
* Projg = norm of the final projected gradient
* F = final function value
* \* \* \*
* N Tit Tnf Tnint Skip Nact Projg F
* 6 42 59 1 0 0 1.788D-05 8.838D+00
* F = 8.8379038638630973
* CONVERGENCE: REL\_REDUCTION\_OF\_F\_<=\_FACTR\*EPSMCH
* [82]:

|  |  |  |  |
| --- | --- | --- | --- |
| SARIMAX Results | | | |
| **Dep. Variable:** | COMED\_MW | **No. Observations:** | 4748 |
| **Model:** | SARIMAX(1, 0, 1)x(2, 1, 1, 7) | **Log Likelihood** | -41962.368 |
| **Date:** | Mon, 11 Dec 2023 | **AIC** | 83936.735 |
| **Time:** | 17:42:15 | **BIC** | 83975.500 |
| **Sample:** | 01-01-2002 | **HQIC** | 83950.360 |
|  | - 12-31-2014 |  |  |
| **Covariance Type:** | opg |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **coef** | **std err** | **z** | **P>|z|** | **[0.025** | **0.975]** |
| **ar.L1** | 0.8329 | 0.007 | 115.306 | 0.000 | 0.819 | 0.847 |
| **ma.L1** | 0.3704 | 0.012 | 31.986 | 0.000 | 0.348 | 0.393 |
| **ar.S.L7** | 0.0611 | 0.012 | 4.924 | 0.000 | 0.037 | 0.085 |
| **ar.S.L14** | 0.0566 | 0.013 | 4.374 | 0.000 | 0.031 | 0.082 |
| **ma.S.L7** | -0.9917 | 0.002 | -402.031 | 0.000 | -0.997 | -0.987 |
| **sigma2** | 2.933e+06 | 4.29e+04 | 68.427 | 0.000 | 2.85e+06 | 3.02e+06 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Ljung-Box (L1) (Q):** | 1.20 | **Jarque-Bera (JB):** | 789.25 |
| **Prob(Q):** | 0.27 | **Prob(JB):** | 0.00 |
| **Heteroskedasticity (H):** | 1.08 | **Skew:** | 0.23 |
| **Prob(H) (two-sided):** | 0.13 | **Kurtosis:** | 4.95 |

# Predictions were made on the test data, and the forecast was plotted against the actual values to visually assess the model's performance. The Root Mean Squared Error (RMSE) was computed, resulting in a value of 4681.8977, which measures the average magnitude of the forecast errors.

# A graph with blue lines and a red line Description automatically generated

# A white rectangular object with red text Description automatically generated

# A screenshot of a computer Description automatically generated

# USING PROPHET Algorithm (Please refer my IPYNB Code notebook for detailed workflow and Explanation):

# The process remains the same as we have done until Model building in SARIMA.

# A graph of data with numbers and text Description automatically generated with medium confidence

# Comparison of Models:

# print(f"ARIMA RMSE: {arima\_rmse:.2f}")

# print(f"Prophet RMSE: {prophet\_rmse:.2f}")

# ARIMA RMSE: 4681.90

# Prophet RMSE: 3269.18

# Takeaways

# ARIMA (Autoregressive Integrated Moving Average):

# ARIMA models are a staple in time series forecasting. They are designed to capture autocorrelation in the data, or in other words, the idea that future data points are related to past ones.

# Advantages:

# simple and interpretable parameters

# less tuning needed compared to machine learning models

# Disadvantages:

# requires stationary data

# struggles with multiple seasonal patterns

# computationally intensive for large datasets

# Prophet:

# Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data.

# Advantages:

# handles multiple seasonalities well

# flexible with inclusion of holiday effects and additional regressors

# easy to use (does not require a deep understanding of its underlying implementations)

# Disadvantages:

# components of the forecast are not easily interpretable as those in ARIMA

# less Effective on High Frequency Data (e.g., data collected every minute or every second)