Summary Report

Jin Qin

May 2024

1 Assignment 1

1.1 Analysis

To start addressing the problem, we first need to understand the definitions of all ISOs based on official documents and web research. The definitions are as follows:

- **PJM Onpeak(16 hrs)**: 7 am to 11 pm on Mondays through Fridays, except holidays as defined in the PJM Manuals.
- **MISO Onpeak (16 hrs)**: 6 am to 22 pm weekdays: 16hrs excluding NERC holidays or if the holiday occurs on a Sunday, the Monday immediately following the holiday.
- **ERCOT Onpeak (16 hrs)**: weekdays from ending 7:00 am to 10:00 pm excluding NERC holidays.
- **SPP Onpeak (16 hrs)**: 6 am to 22 pm excluding Saturdays and Sundays and all NERC holidays
- **NYISO Onpeak (16 hrs)**: 7:00 am to 11:00 pm Monday through Friday, except for NERC-defined holidays
- **WECC Onpeak (16 hrs)**: 6:00 am to 10:00 pm CDT on weekdays, excluding NERC holidays
- **CAISO Onpeak(16 hrs)**: Monday through Saturday, 6:00 am to 22:00 pm, Pacific Prevailing Time (PPT), excluding NERC Holidays

Where the special cases in the ISO's are MISO and CAISO. In MISO, if a NERC holiday occurs on a Sunday, the following Monday is considered off-peak. In CAISO, on-peak includes six days (Monday through Saturday), unlike others that typically cover five weekdays. The NERC holidays are:

- **New Year's Day**: January 1
- **Memorial Day**: last Monday of May

- **Fourth of July**
- **Labor Day**: first Monday of September
- **Thanksgiving**: fourth Thursday of November
- **Christmas**: December 25

We also need to consider daylight saving in calculating the on-peak. With these definitions clarified, we can proceed with the algorithm and coding.

1.2 Approach

- **get_dst_changes(year)**

This function takes in a specific year and returns the start date and end date of the daylight saving period, typically starting on the second Sunday in March (between March 8 to March 15) and ending on the first Sunday in November (between November 1 to November 8).

- **get_exclude_dates(year, iso, if_2x16=False)**

This function takes in a specific year and the desired ISO. The **if_2x16** parameter indicates the special case in MISO where the 2x16 calculation applies. This function returns a list of excluded dates for the specified year, considering the holidays for other ISOs and the Monday following a holiday in MISO.

- **parse_input(period)**

This function identifies the input character and calculates the start date and end date for the desired period. If the input string contains 'Q', it identifies the period as a quarter of a year. If the string ends with 'A', it identifies the period as annual (January 1 to December 31). If the string length is 7, it identifies the period as year + month abbreviation (e.g., 2018MAR). If the input string contains a dash, it identifies the period as a specific date (e.g., 2018-03-31). If none of these conditions match, the function raises an exception.

- **calculate_peak_hours(start_date, end_date, iso)**

This function takes in the start date, end date, and ISO, and calculates the **on-peak hours, off-peak hours, flat hours, 7x8 hours, and 2x16H hours** day by day.

For weekdays: If ISO is MISO, on-peak hours increase by 16, off-peak hours by 8, flat hours by 24, and 7x8 hours by 8 without considering DST. For other ISOs, on a normal weekday, peak hours increase by 16, off-peak hours by 8, flat hours by 24, and 7x8 hours by 8. On the start date of DST, peak hours increase by 15, off-peak hours by 8, flat hours by 23, and 7x8 hours by 8. On the end date of DST, peak hours increase by 17, off-peak hours by 8, flat hours by 25,

and 7x8 hours by 8.

For weekends and holidays: If ISO is MISO, off-peak hours increase by 24, flat hours by 24, and 7x8 hours by 8. For other ISOs, on a normal weekend or holiday, off-peak hours increase by 24, flat hours by 24, and 7x8 hours by 8. On the start date of DST, off-peak hours increase by 23, flat hours by 23, and 7x8 hours by 8. On the end date of DST, off-peak hours increase by 25, flat hours by 25, and 7x8 hours by 8.

For 2x16H calculations: On a normal weekend or holiday, 2x16H increases by 16. On the start date of DST, 2x16H increases by 15. On the end date of DST, 2x16H increases by 17.

- **get_hours(iso, peak_type, period)**

This function takes in the ISO, peak type, and a string representing the period. It returns the ISO, peak type, start date, end date, and the desired hours. The function first checks if the ISO is in the ISO list and if the peak type is in the peak type list. It calculates the start and end date using **parse_input**, then calculates all types of hours using **calculate_peak_hours**, and returns the desired information.

1.3 Evaluation

Based on a manual review, we can obtain the following examples:

```
# Example usage
print(get_hours("MISO", "onpeak", "2018-2-3"))   # Daily
print(get_hours("PJM", "flat", "2018Mar"))    # Monthly
print(get_hours("PJM", "2x16H", "2018Mar"))    # Monthly
print(get_hours("PJM", "7x8", "2018Mar"))    # Monthly
print(get_hours("CAISO", "flat", "2018Mar"))    # Monthly
print(get_hours("CAISO", "offpeak", "2018Q2"))     # Quarterly
print(get_hours("MISO", "onpeak", "2018A"))      # Annually
```

```
{'iso': 'MISO', 'peak_type': 'ONPEAK', 'startdate': '2018-02-03', 'enddate': '2018-02-03', 'num_hours': 0}
{'iso': 'PJM', 'peak_type': 'FLAT', 'startdate': '2018-03-01', 'enddate': '2018-03-31', 'num_hours': 743}
{'iso': 'PJM', 'peak_type': '2X16H', 'startdate': '2018-03-01', 'enddate': '2018-03-31', 'num_hours': 143}
{'iso': 'PJM', 'peak_type': '7X8', 'startdate': '2018-03-01', 'enddate': '2018-03-31', 'num_hours': 248}
{'iso': 'CAISO', 'peak_type': 'FLAT', 'startdate': '2018-03-01', 'enddate': '2018-03-31', 'num_hours': 743}
{'iso': 'CAISO', 'peak_type': 'OFFPEAK', 'startdate': '2018-04-01', 'enddate': '2018-06-30', 'num_hours': 952}
{'iso': 'MISO', 'peak_type': 'ONPEAK', 'startdate': '2018-01-01', 'enddate': '2018-12-31', 'num_hours': 4176}
```

Figure 1: Useage Examples

2 Assignment 2

2.1 Analysis

To create reusable code for this problem, we need to develop functions that perform the following tasks:

1. Identify the time column

2. Identify the index column
3. Identify the content column
4. Resample the time column to hourly intervals.
5. While merging two data frames on the time column convert power consumption values to kilowatts (KW) and calculate the total power consumption by summing the row values.

2.2 Approach

- **detect_and_normalize_timestamp(text)**

This function takes in a string and detects if the string is a date or not. The function can detect date strings with the pattern: month/day/year or month/day, and time strings with the format: hour:minute:second or hour:minute. Upon detection, it returns the standardized date and time string.

- **find_and_normalize_timestamp_column(df)**

This function takes in a data frame and detects and normalizes the time column using **detect_and_normalize_timestamp**. It converts the time column into datetime format, then returns the index of the time column and the normalized data frame.

- **detect_index_column(df)**

This function takes in a data frame and detects columns with unique sequential numbers. If a column is identified with unique sequential numbers, then it is an index column. The function returns the index of the index column.

- **parse_df(df)**

In this function, we first detect the index column using **detect_index_column(df)**. If the data frame has an index column, we remove this column. Then, we detect and normalize the time column using **find_and_normalize_timestamp_column(df)**. For the columns that are neither time nor index columns, we name them content columns. The function returns the index of each type of column and the normalized data frame.

- **resample_and_merge(hourly_data, idx1, minute_data, idx2)**

This function takes in the hourly data and the minute data, along with the indices for different columns in each data frame in the order [time_column, content_column]. It resamples the minute data by summing to an hour, performs an inner join on the data frames, and calculates the power consumption in KW. It then creates a new column named **Total_Hourly_Consumption_KW** by summing the rows in the merged data frame.

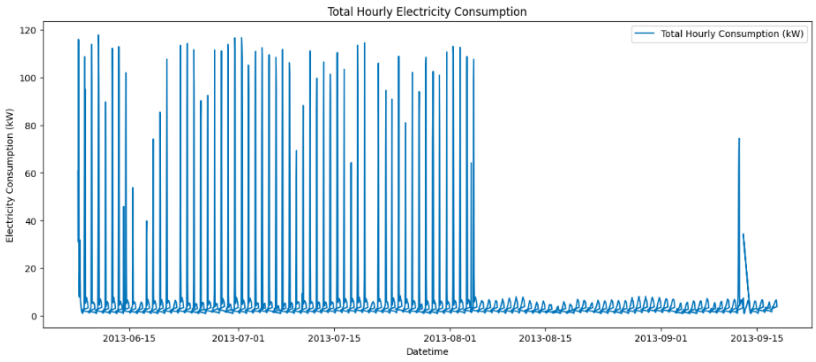


Figure 2: Hourly Power Consumption

2.3 Visualization

First, we plotted a line chart to observe the trends in hourly power consumption (Figure 2). It is evident that the new appliance ceased functioning during August and early September. Notably, the power consumption of this new appliance was significantly higher than the total power consumption of all other appliances in the entire room, which is abnormal.

After grouping the data by month (Figure 3), we can observe the trend more clearly. The power consumption in August and September is considerably lower than the power consumption in June and July. Grouping the data by weekdays

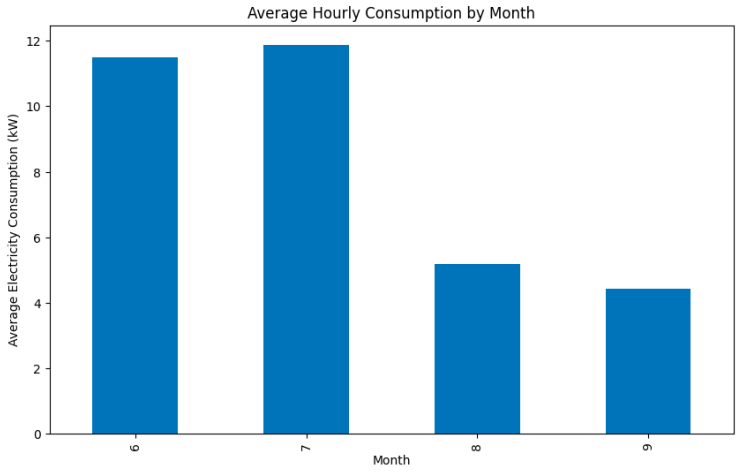


Figure 3: Power Consumption Grouped by Month

does not reveal significant information (Figure 4). It appears that Tuesday

and Wednesday have the lowest power consumption. However, we cannot draw substantial inferences from the current data.

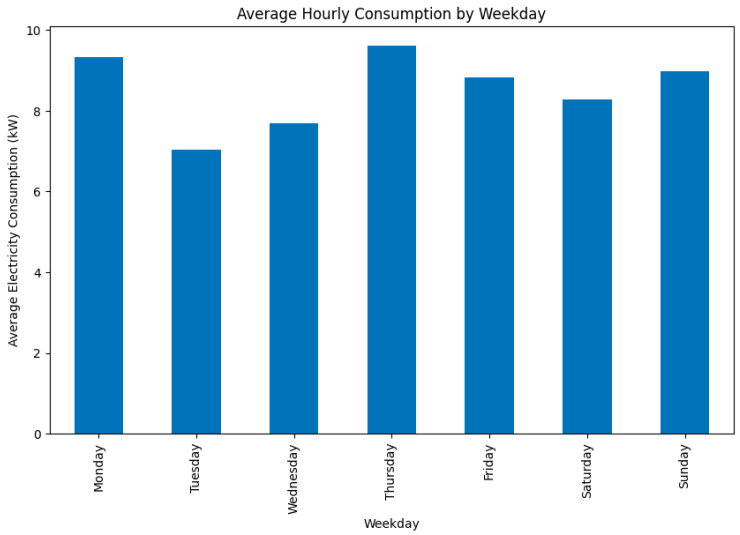
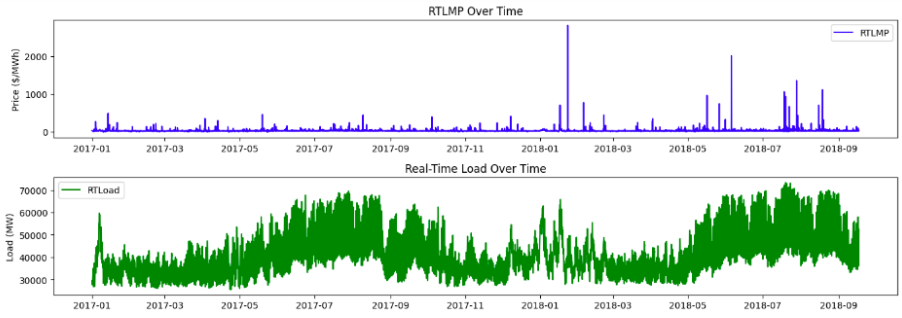


Figure 4: Power Consumption Grouped by Weekdays

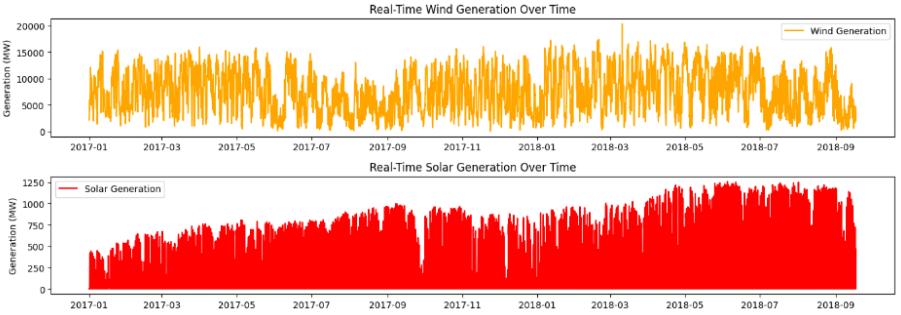
3 Assignment 3

By summarizing the statistics of the meaningful columns (excluding time columns) in the given time series, we arrive at Table 1. We observe that the data in all columns have a large range. For the column we want to predict, namely RTLMP, the mean is 25.8, with a maximum of 2809.4 and a minimum of -17.9. This indicates the presence of outliers in this data. The mean values of all other meaningful data are significantly larger than that of RTLMP. However, the scale does not matter when using ARMAV. It is important to note that there are missing values in the data, which should be replaced with 0 or the mean value between the preceding and succeeding data points.

With all the metrics known, we should move on to visualization. From Figure 5, we can clearly observe a stochastic trend in **WIND_RTI** and a deterministic trend in **RTLOAD**. For **GENERATION_SOLAR_RT**, the trend is not obvious from visualization; we may assume it has a deterministic trend. As for **RTLMP**, the data contains many outliers, making visualization challenging. Therefore, it is difficult to observe a trend using a line chart for this data.



(a) RTLMP and WIND_RTI



(b) GENERATION_SOLAR_RT and RTLOAD

Figure 5: Line Chart of all Data

Metrics	RTLMP	WIND_RTI	GENERATION_SOLAR_RT	RTLOAD
Count	14987	14982	14983	14987
Mean	25.8	7532.4	2912.0	42371.7
Min	-17.9	54.4	0	25566.5
Max	2809.4	20350.4	1257.5	73264.7
Std	46.4	3992.9	370.9	987.4

Table 1: Descriptive Statistics for HB_NORTH (RTLMP), ERCOT (WIND_RTI), ERCOT (GENERATION_SOLAR_RT), and ERCOT (RTLOAD)

Now, we should turn to the heatmap of correlation coefficients between each column (Figure 6). The only positive correlation we can confirm is between **GENERATION_SOLAR_RT** and **RTLOAD**. Since we plan on using an ARMAV model for the data, multicollinearity can be problematic. Highly collinear time series can lead to unstable estimates of the coefficients and inflate the standard errors, making the model's predictions less reliable. With correlation coefficients around 0 for most of the columns, we can confirm that there is hardly any multicollinearity in the data. This indicates that we are in a good position to proceed with modeling using the current data.

3.1 ARMAV Modeling

In this section, we aim to predict **RTLMP** with **WIND_RTI**, **GENERATION_SOLAR_RT**, and **RTLOAD** The input data utilized in the modeling process include:

$$\vec{X_t} = \{X_{1t}, X_{2t}, X_{3t}, X_{4t}\}$$

- X_{1t} : **WIND_RTI**
- X_{2t} : **GENERATION_SOLAR_RT**
- X_{3t} : **RTLOAD**
- X_{4t} : **RTLMP**

The leading indicators, denoted as X_{1t}, X_{2t}, X_{3t}, are used to predict the output data, which is the one-step-ahead forecast of **RTLMP**. By constructing the model in Python and using the Akaike Information Criterion (AIC) to select the best model, we achieved a Mean Squared Error (MSE) of approximately 3915.14. This result is not entirely surprising given the presence of numerous outliers in the data and considering that the ARMAV model essentially functions

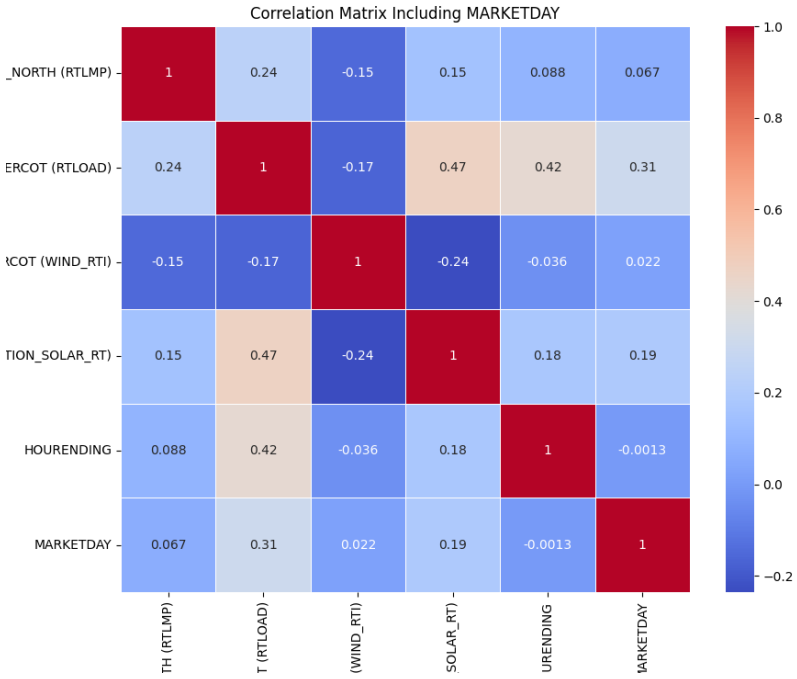
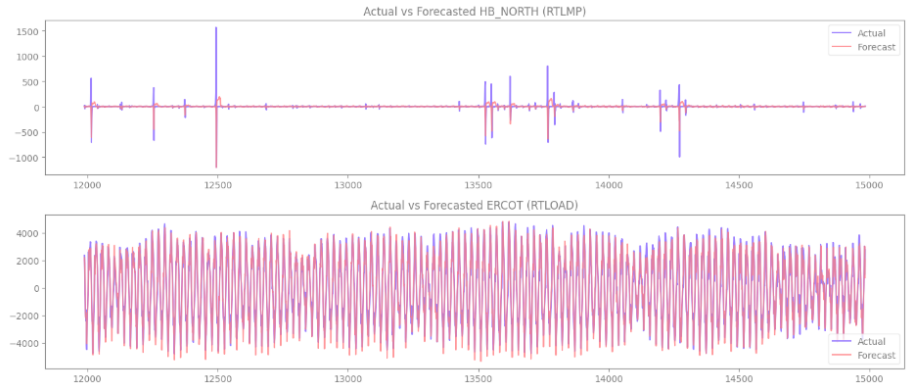


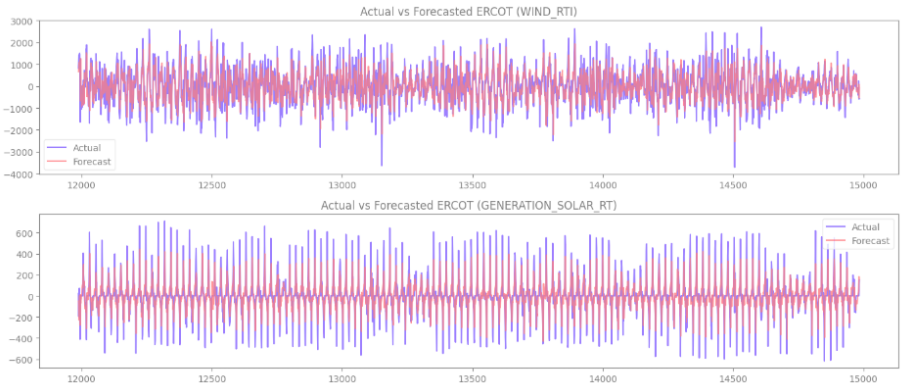
Figure 6: Correlation Coefficient of all columns

as an ARV model in Python without the moving average component, we are not able to capture the information in the residuals. Nonetheless, this performance is an improvement over the ARMA model (with RSS around 5900), which relies solely on **RTLMP** to predict itself (I didn't include the ARMA model in my code or report because it didn't go well).
At this point, I must acknowledge that I did not accomplish this assignment as well as I had hoped. My unfamiliarity with ARIMA/SARIMAX modeling in Python posed a challenge, as Python is not the optimal tool for handling complex, high-order ARMAV models due to its limited computational capabilities. Python can manage ARMA models up to an order of 15 (ARMA(15,14)), but for higher-order models and faster computation, MATLAB is a superior tool. MATLAB's ability to efficiently manage higher-order models makes it more suitable for advanced time series analysis.

In MATLAB, we can detect seasonality by analyzing the roots of the AR part and fitting parsimonious models using the F-criterion. My professor has developed a tool that automatically determines the best order of ARMA in MATLAB by embedding a F-test within the function, hence there's no need for grid search-



(a) ARMAV prediction of RTLMP and WIND_RTI



(b) ARMAV prediction of GENERATION_SOLAR_RT and RTLOAD

Figure 7: Line Chart of all Data

```
Mean Squared Error for HB_NORTH (RTLMP): 3915.1241500898636
Mean Squared Error for ERCOT (RTLOAD): 273426.79178864
Mean Squared Error for ERCOT (WIND_RTI): 282886.7663566297
Mean Squared Error for ERCOT (GENERATION_SOLAR_RT): 10646.321204670534
```

Figure 8: MSE of ARMAV model

ing. Using this tool, I completed a time series modeling project on Apple stock prices, achieving a residual sum of squares (RSS) of 36.5. I have included a graph of my predictions along with a 95% confidence interval for your review. Please feel free to contact me if you are interested in using MATLAB for prediction!

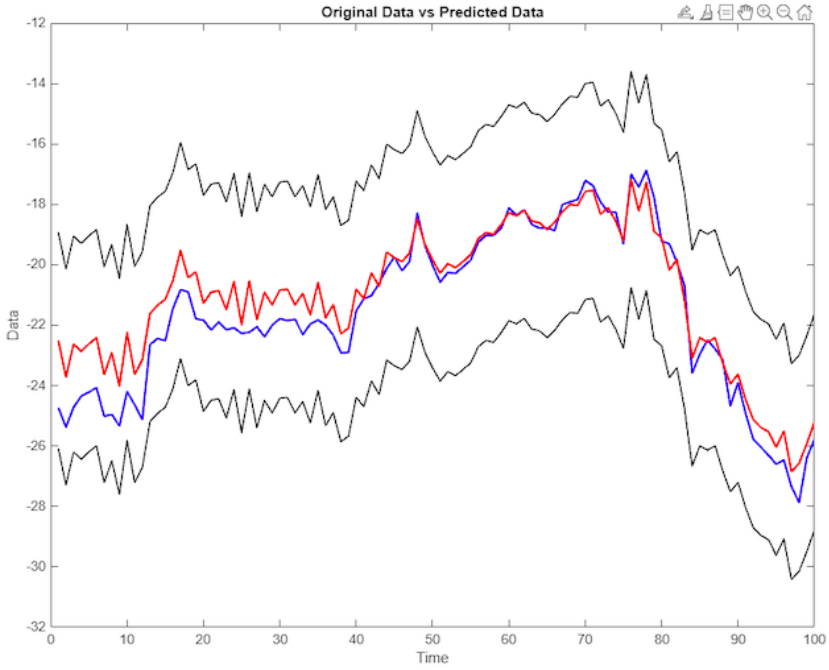


Figure 9: One-step-ahead prediction of ARMAV(4,3) with 95% CI on detrended data