Stephen Tambussi

Dylan Hoover

ELEN 285

Introduction to the Smart Grid Project Paper

This paper will analyze the electrical load of commercial sites participating in demand response (DR) services, a core component of the Smart Grid. Two commercial sites have been selected for analysis from an anonymized EnerNOC database (1): site 690 and site 697. Site 690 is based in New York state (NY), near the city of Jamestown, and is classified as light industrial manufacturing (8). Site 697 is based near San Jose, California and is classified as light industrial food processing (15). For each site, the ISO and utility with their associated DR programs will be detailed first and then each site will be analyzed to discern its load behavior and participated DR events.

Site 690 Analysis

Site 690 is located in NY state close to Jamestown which is serviced by New York ISO (NYISO). NYISO covers the entire state of NY and places this site in zone A (2). The electric utility for this site is most likely Jamestown Board of Public Utilities (BPU) (9)(10). However, it is not possible to pinpoint the exact electric utility due to the anonymized location data of each site. NYISO provides multiple DR programs and since the utility cannot be determined the focus will be on NYISO's DR programs.

The DR programs offered by NYISO are broken down into two categories, reliability-based programs and economic-based programs (3). In reliability-based programs, NYISO determines the activation of a DR event for the purpose of reducing load for a specified amount of time, to supplement generation when operating reserves are predicted to be short, or when there is an actual operating reserve deficiency or another system emergency (4). The reliability-based programs are the Emergency Demand Response Program (EDRP) and ICAP-Special Case Resources (SCR) (3). In economic-based programs, resources determine when to participate in DR events. The purpose of these programs is to reduce load and compete with electric generation (4). DR events for these programs are scheduled by NYISO based on economic offers. The economic-based programs are the Day-Ahead Demand Response Program (DADRP) and Demand-Side Ancillary Service Program (DSASP) (4).

Any of these programs are applicable to site 690 because of the generic nature of the events. The main concern would be that some of the programs such as EDRP and SCR require a minimum of 100 kW decrease in energy consumption (4). However, site 690 uses far more than 100 kW which allows it to be eligible for these programs. NYISO has three types of baselines, Average Coincident Load (ACL), Customer Baseline Load (CBL), and Real-Time Baseline. ACL is used for the SCR for capacity auctions program (4). The programs EDRP, SCR Energy, and DADRP use CBL for their baseline (4). Finally, DSASP uses the Real-Time baseline (4). For the computation of ACL, the reference period is any prior equivalent capability period and it is calculated from the average of the highest 20 resource loads during the top 40 NYCA peak load hours in the same season (Summer/Winter) of the previous year (4). CBL's computation has a reference period of the five highest consumption days of the last 10 "like" days where DR events did not occur, with the option for a weather-sensitive adjustment (4). The Real-Time baseline computation uses the actual load just prior to the beginning of a real-time schedule (4).

The baseline computation approaches described above each have their limitations in terms of accuracy and fairness. Average Coincident Load uses data from the same seasons of the prior year to determine the baseline. However, the seasonal weather each year can vary significantly which would change the highest 20 resource loads and affect the baseline for the following year. Customer Baseline Load seems to be the most fair in terms of its baseline calculation. But, a consumer with knowledge of the CBL calculation and the expected DR event day could manipulate the calculation by maintaining high loads for the 10 eligible days, thereby artificially inflating their baseline and allowing them to keep a higher load during the event. Finally, the real-time baseline could be the easiest to manipulate. If NYISO announces the real-time schedule well before it begins, then a consumer could change their load prior to the start of the schedule to manipulate their baseline.

Classified as light industrial manufacturing, site 690 could be visualized as a factory with heavy machinery that draws a lot of power. However, it is difficult to completely discern this information as each site is anonymized to protect EnerNOC customer data. The dataset for site 690 is a CSV file containing a table composed of five data columns: timestamp, dttm_utc, value, estimated, and anomaly. Timestamp data is an increasing integer value, dttm_utc data is the date and time in coordinated universal time, value data is the power load reading in kW/h, estimated data is a boolean value representing if the reading was estimated, and anomaly data is any non-blank value if the reading was erroneous. Each row in the table is a 5 minute interval containing values for the previously mentioned columns, with the total number of rows being 104,857. Every data point with value readings of 0 kW/h or an anomaly indicator of non-blank was removed from the site analysis. Ultimately, only 12 data points were removed, with the total percentage of missing data being approximately 0.0114%. The data analysis and plotting was performed with the use of Python and the libraries NumPy, Pandas, and Matplotlib. The relevant code is located in appendix A.

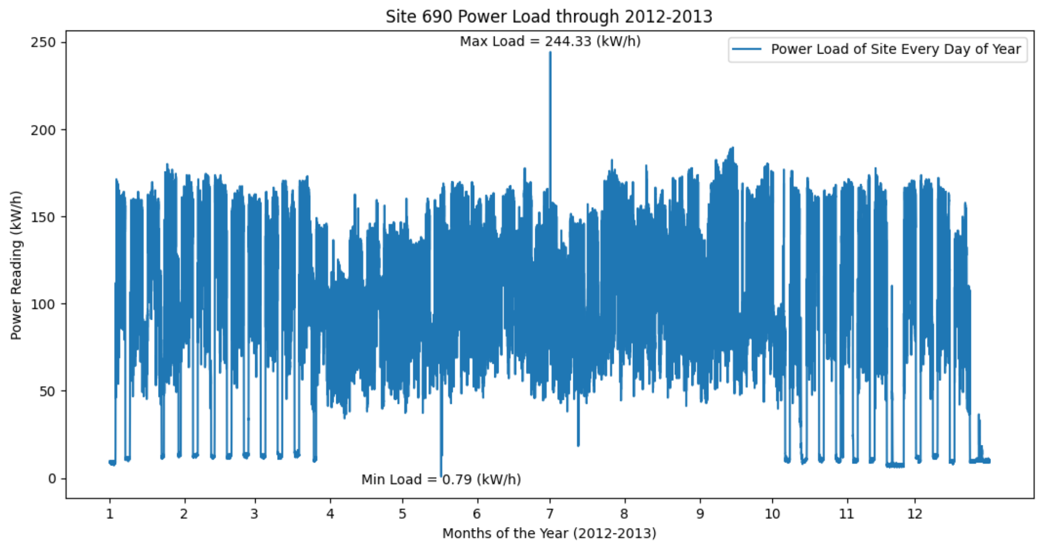

Figure 1: Power Load of Site 690 through 2012-2013

The above figure displays the power load of site 690 in 5 minute intervals throughout the entire year. The site has fairly consistent maximum loads (between 150 and 200 kW/h) throughout the months, but with a greater variation in minimum loads. There are some outliers in the readings with the highest recorded load at 244.33 kW/h and the lowest recorded load at 0.79 kW/h. Furthermore, figure 1 displays more variation in power use of the site for the months of January through March and October through December. Specifically, the minimum loads are lower during these months. Only during the spring and summer months, April through September, does the minimum load remain consistently higher. This variation could be attributed to the increased temperatures of spring and summer with the site requiring continuous cooling even outside operating hours.

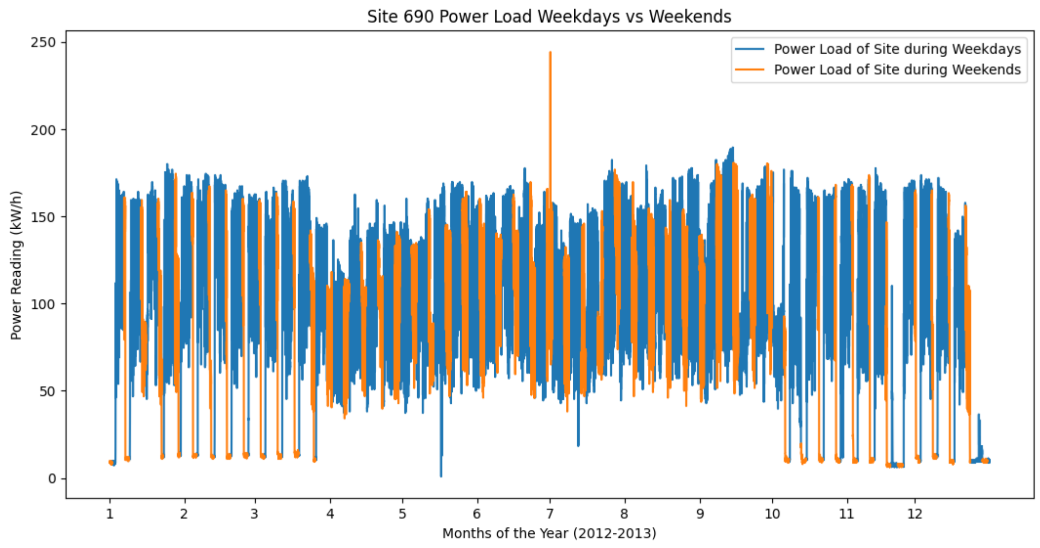

Figure 2: Weekdays vs Weekends Power Load of Site 690

As shown in the above figure, site 690 has consistently lower maximum loads on the weekends compared to on the weekdays. The minimum loads are closer in value regardless of the day being a weekday or weekend. The aforementioned outliers, lowest recorded load and highest recorded load, occurred on a weekday and weekend respectively. Moreover, the lower maximum loads on the weekends displayed in figure 2 could correlate to a reduced manufacturing output for the site. With the assumption that the site is a kind of factory, a reduced manufacturing output and therefore power load on the weekends would be reasonable considering that a typical American business operates 9AM to 5PM, Monday through Friday.

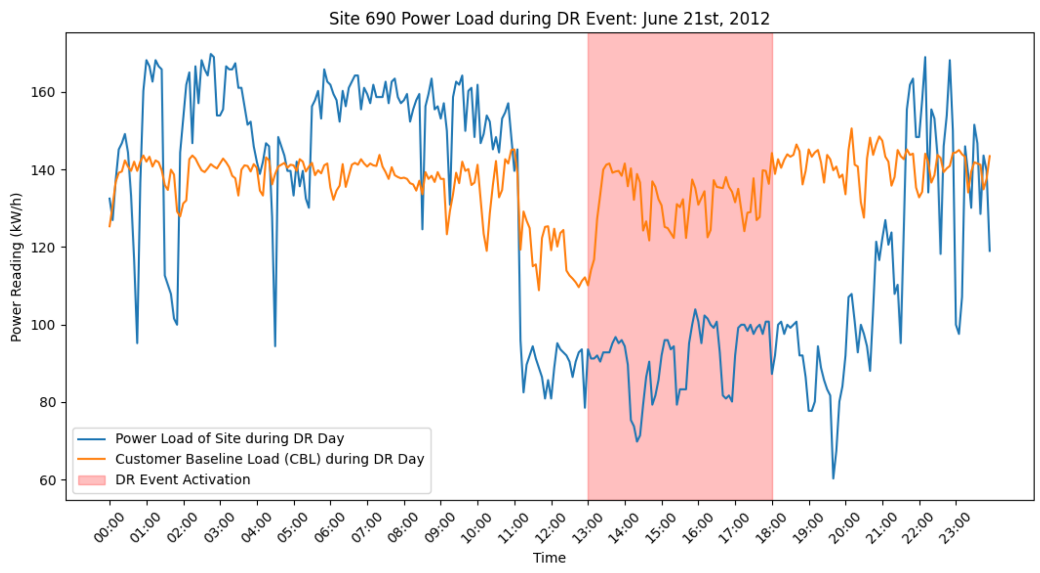

Figure 3: DR Event Power Load of Site 690

Figure 3 displays the power load readings for site 690 during the DR event day along with the calculated baseline of which the site's load reduction performance is measured. The graph illustrates that the site did participate in the DR event for its entire duration. As previously mentioned, this site is serviced by NYISO and they were responsible for calling the event on that day, June 21st, 2012 (6). The event window was 1PM to 6PM and the DR event program was SCR/EDRP (6). This program is reliability-based and is only called when there is a predicted/actual reserve deficit or another system emergency. The cause for this event was most likely due to the incredibly hot temperature in the site area on that day. With a maximum temperature between 90-100℉, the DR event corresponds perfectly to the expected temperature levels throughout the day as at 1PM (the start of the event) the temperature would approach its maximum and at 6PM (the end of the event) the temperature would decrease (7). Therefore, the likely reason behind NYISO calling the event was to reduce the demand on the electrical grid from the wide-spread usage of energy-intensive cooling systems.

The baseline in figure 3 is CBL which corresponds to the SCR/EDRP baseline method for NYISO. The computation of this baseline was automated through the Python program in appendix A and created with the *SPECIFICATION EE-DR-01* document from Con Edison on general CBL calculation methods (5). All other information on this baseline method was detailed previously. Figure 3 illustrates that site 690 was capable of reducing their load from ~170 kW/h to ~100 kW/h for the event. Their baseline indicates that the load was expected to be between ~140 to ~150 kW/h. Overall, this site was able to remain under the baseline for the entire duration of the event and only increased their load well after the event had ended.

Site 697 Analysis

Site 697 is a small light industrial site that focuses on food production. The site is very small compared to others at 4650 square feet, this site is located near the bay area as it is serviced by PG&E and the coordinates provided by the datasheet place it near the Santa Cruz mountains. As previously mentioned, this site is serviced by PG&E and CAISO (16), with PG&E triggering the chosen DR event. The DR programs available from PG&E are peak day pricing, base interruptible program, capacity bidding program and emergency load reduction program (12). There are also private companies that offer third party programs that are contracted by PG&E. The baselines for these programs are slightly different from those used by other programs provided by NYISO for site 690. For peak day pricing, the baseline is a time-of-use rate where the participants pay higher energy rates during critical peaks. This means there is no real calculation, but the utility decides what to charge during peaks. Base interruptible program participants agree to drop load to a firm service level also set by the utility or ISO. The standard baseline used for other programs is a 3-in-10 baseline where the average of the three highest energy usage days in the immediate past ten days is used to calculate a baseline (14). This site has a very low percentage of unused data points with a percentage of 0.0076% missing.

Analyzing the baseline calculations used for these different programs, the 3-in-10 is useful for sites where the load is consistent throughout the entire year or at least consistent during the time period being used. However, as discussed later, the load for this site is not very consistent, even from week to week, making it difficult to use the past 10 days as a baseline. The weaknesses in the other baselines used is that they don't really utilize a calculation when determining how much load should be reduced by because it is an overarching payment for being during peak demand or just a flat rate that the grid needs to be reduced by. This approach is useful for when the sites themselves have aggregators that help with auto demand response but not very useful if a site doesn't keep track of their baseline.

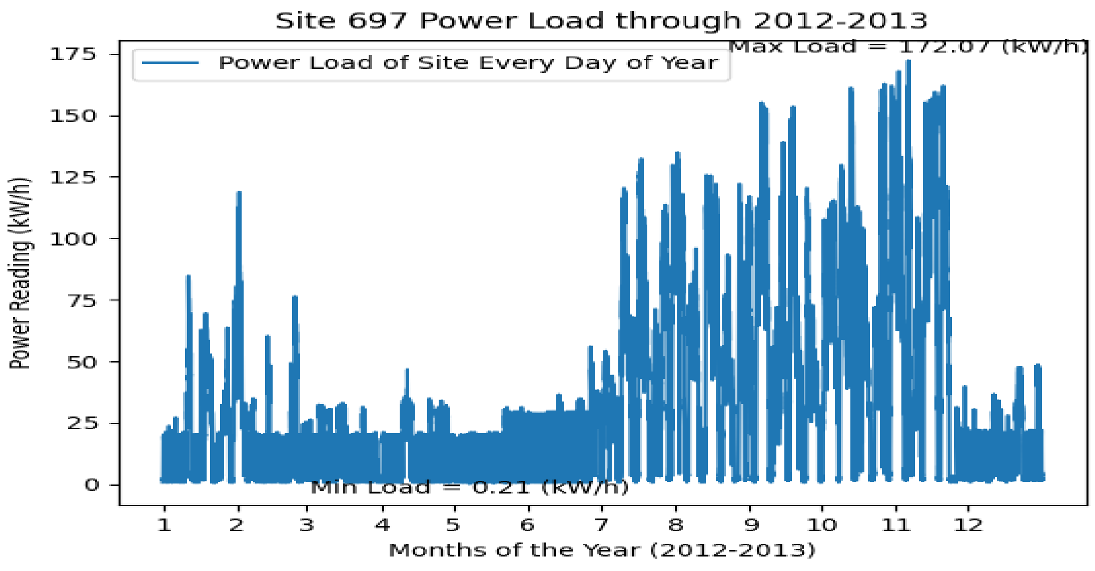

Figure 4: Power Load of Site 697 through 2012-2013

The figure above displays the power load of site 697 throughout the year 2012. The minimum load is 0.21 kW/h and the maximum load is 172.07 kW/h. Something interesting about the graph is the higher power usage in the later half of the year. Based on the location of the plant, this could be based on weather, seasonal use, or what type of food production is done at the plant. It is more than likely due to the change in weather patterns resulting in the need to heat the facilities during the cold months. Another interesting thing to note is how variable the maximum and minimum power load is during the colder months opposed to the possible off season of the factory where the maximum and minimum values are very consistent for a few months. One additional point to note is that the minimum load, 0.21 kW/h, is reached throughout the entire year, not just during the lower power draw period. Whereas the maximum load is only reached during the time period where the peak power load average is around 140 kW/h.

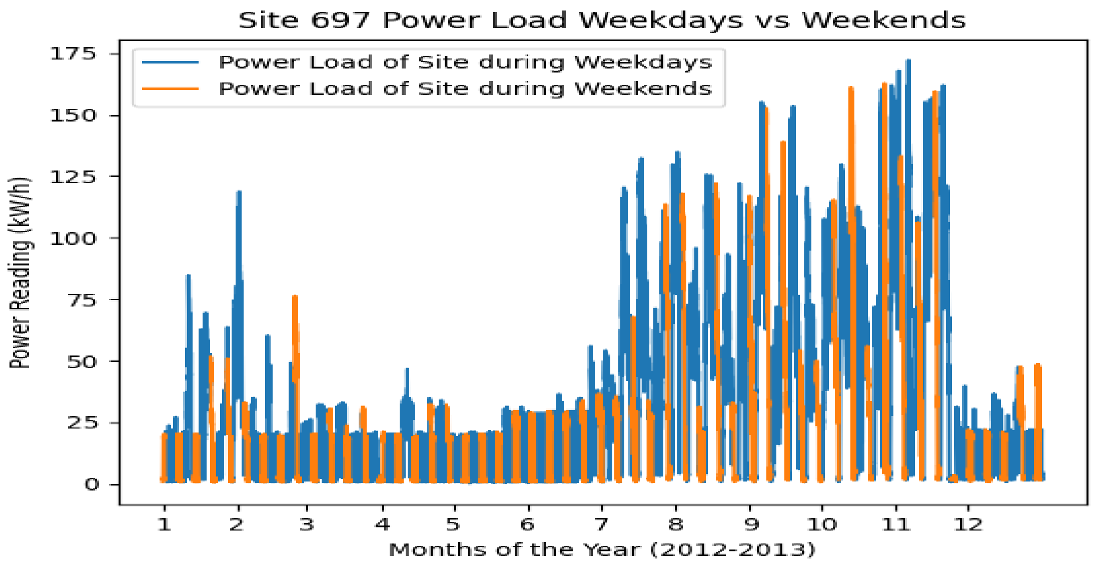

Figure 5: Weekdays vs Weekends Power Load of Site 697

The figure above displays the power of the site of the year for both the weekdays and the weekends. Throughout the low periods of power load the weekends and weekdays are very consistent, likely hinting that this is just the base load it takes to keep this site powered on during the off season for it. This is also true for the periods where the load is much higher. There are some outliers where weekend use is higher than weekday use, but this could be attributed to the plant trying to reach a deadline over the weekend or inconsistent working patterns.

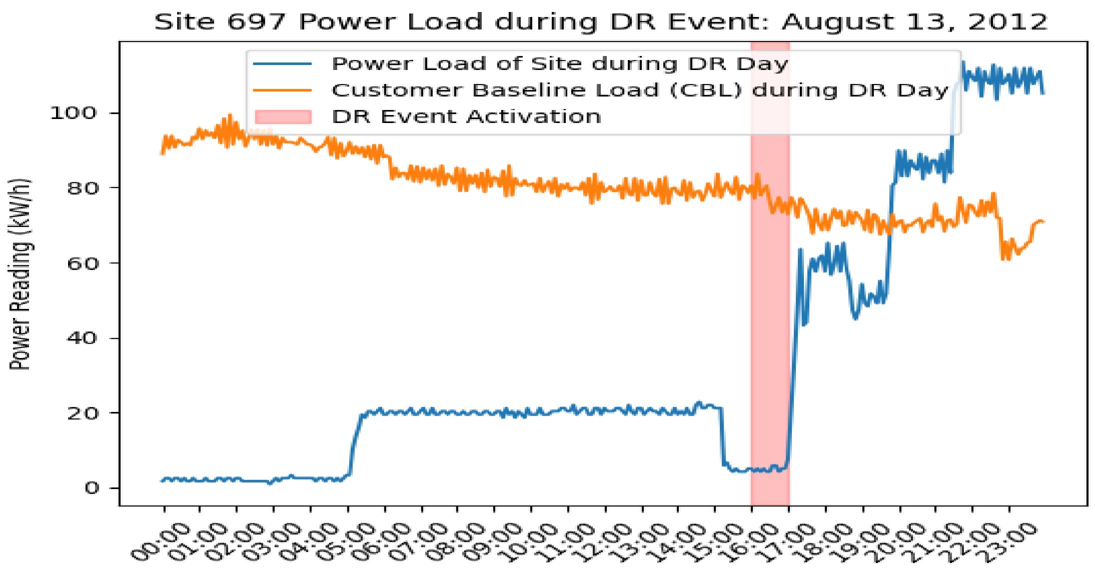

Figure 6: DR Event Power Load of Site 697

The figure above displays a time when this site participated in a demand response event on August 13, 2012 (11). Specifically, the event was for a capacity bidding program. It was a day of trigger based on the heat rate with the temperature at a high of 84 °F throughout the event (13). This event only lasted an hour between 4pm and 5pm, but the site ramped down their power usage an hour previous, possibly due to an automatic demand response. A maximum 19.4 MW load reduction was needed during this event which was triggered by PG&E. For this type of event there was no minimum demand load requirement. An interesting note about the capacity bidding program is that sites can have third party aggregators handle their automated demand response. As mentioned previously, this could be why the site ramped down power before the event trigger.

Looking at the calculated 3-in-10 baseline for site 697 and utilizing the past 10 days where a DR event was not called, the plot doesn't quite fit the correct power reading of this site during the day of the demand response event. This can be attributed to the high variability of the site's power usage during the week. A reason for why the site chose to participate in this specific

DR event could be because there was less work going on in the earlier parts of the day. Therefore, allowing them to participate in this event, which could then explain the instantaneous ramp up in power usage after the event for use during work later in the night. It should be noted that although the baseline curve does not line up in terms of power usage levels, it does create an accurate representation of what the power chart would have looked like if the DR event was not called.

DR Event Algorithm Design and Conclusions

Designing an algorithm to automatically detect the occurrence of a DR event would require many inputs. Without access to a utility's or ISO's announcement of DR events, the algorithm would have to analyze the load behavior of a targeted site. First, this algorithm would have to learn the scheduled fluctuations of a site's power load such as shutting off machines, lights, etc. at the end of a work day or the low load resulting from non-operating days such as holidays and weekends to be able to ignore the typical load reduction behaviors. Learning these behaviors would necessitate the integration of a machine learning model that requires large amounts of data to train on. Optionally, the algorithm could take into consideration the season of the year. While a DR event can occur at any time of the year, they are more likely to occur during the summer and winter months due to high usage of HVAC systems. This data could aid the algorithm to be more accurate in its predictions. However, it would also need to take into account the location of the site due to seasonal weather variations at different locations. Finally, the most important data the algorithm requires is the expected baseline so that it can identify a sudden reduction in load from it. Therefore, this algorithm would analyze the load values of the site to recognize a sustained reduction that is outside the learned variations to determine a DR event occurrence.

The challenges of designing this algorithm are mainly the uncertainty of event occurrence and the volume of data to process. Even with all the load behavior data, it is still difficult to accurately predict the occurrence of a DR event without official notification from an ISO or utility. There could be other variables that affect the load behavior of a site which would then trigger a false detection of an event, such as local power failures or temporarily offloading power load to generators and other sources. Furthermore, the amount of data that this algorithm would have to process for a single site is extensive. Learning the scheduled load behavior requires the load data from prior years and detecting the event necessitates parsing the data to determine if the current data coincides with a scheduled behavior. Most high-end computers would struggle to run this algorithm due to the sheer amount of data. Therefore, designing an algorithm to automatically detect a DR event through load behavior is not impossible, but very difficult computationally.

Overall, there are stark differences between the two sites. The most obvious being that they are on opposite sides of the country resulting in dramatically different loads throughout the year. For example, site 690 has a much more constant load during the middle of its yearly cycle, while site 697 has a much higher load throughout the end of the year. These differences can be attributed to the different weather conditions and the dissimilar sub-industry of each site. Conversely, these sites have similar maximum loads which comes from the fact that they are both light industrial sites even though they produce different things. In general, the ISOs have many of the same programs and baseline calculations which could be due to emerging standards or just using the most efficient way to calculate baselines.

References

1. *Open Data*. (n.d.). Open Enernoc Data. https://open-enernoc-data.s3.amazonaws.com/anon/index.html
2. *Real-time Dashboard*. NYISO. (n.d.). https://www.nyiso.com/real-time-dashboard
3. *Demand Response*. (n.d.). NYISO. https://www.nyiso.com/demand-response
4. Kumar, M. S. (2022, February 15). *Demand Response* [Slides]. NYISO. https://www.nyiso.com/documents/20142/3037451/9-Demand-Response.pdf
5. *SPECIFICATION EE-DR-01* (Revision 04). (2021, May). Con Edison. https://cdne-dcxprod-sitecore.azureedge.net/-/media/files/coned/documents/save-energy-money/rebates-incentives-tax-credits/smart-usage-rewards/customer-baseline-load-procedure.pdf?rev=1ba985ffd70049a69825fb20615a8394&hash=D883131A38BB9364DD5FE6D914F15945
6. *Called Events & Tests*. (2022, February). NYISO. https://www.nyiso.com/documents/20142/1401632/EXT-DR-Events-and-Tests-History-With-CPs-Thru%2002-16-2022.pdf/babee9b1-fed9-fb32-7553-5297ee905327
7. *Daily Weather Map - June 21, 2012*. (n.d.). National Weather Service. https://www.wpc.ncep.noaa.gov/dailywxmap/index_20120621.html
8. *42.21, -79*. (n.d.). GPS Coordinates. https://gps-coordinates.org/
9. *Electric | Jamestown BPU*. (n.d.). Jamestown BPU. https://www.jamestownbpu.com/237/Electric
10. *New York Utility Coverage Map & Service Territory*. (n.d.). YSG Solar. https://www.ysgsolar.com/blog/new-york-utility-coverage-map-service-territory-ysg-solar

11. *December 2012 - PG&E, Pacific Gas and Electric*. (n.d.). Retrieved March 9, 2022, from https://www.pge.com/includes/docs/pdfs/mybusiness/energysavingsrebates/demandresponse/cs/December2012_ILPreport.pdf

12. *Business Energy Incentive Programs*. Energy incentive programs. (n.d.). Retrieved March 8, 2022, from https://www.pge.com/en_US/large-business/save-energy-and-money/energy-management-programs/energy-incentives.page

13. *Past weather in San Jose, California, USA - August 2012*. Weather in August 2012 in San Jose, California, USA. (n.d.). Retrieved March 8, 2022, from https://www.timeanddate.com/weather/usa/san-jose/historic?month=8&year=2012

14. *Baselines for retail demand response programs - caiso.com*. (n.d.). Retrieved March 9, 2022, from https://www.caiso.com/Documents/Presentation-Baselines_RetailDemandResponsePrograms.pdf

15. *37.2, -122* (n.d.). GPS Coordinates. https://gps-coordinates.org/

16. *Pacific Gas and Electric Company Service Territory*. PG&E, Pacific Gas and Electric - Gas and power company for California. (n.d.). Retrieved March 8, 2022, from https://www.pge.com/mybusiness/customerservice/otherrequests/treetrimming/territory/

Appendix A

Site 690 Code: data analysis and plotting

```
#ELEN 285 Final Project
#For Site 690
from math import floor
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

df_og = pd.read_csv(r"C:\Users\steve\Desktop\ELEN285\Final
Project\csv\690.csv") #unmodified dataframe
df = df_og.drop(df_og[df_og['value'] == 0].index) #remove all rows with
error values (0 kWh)

#Numpy arrays of each column in the csv
np_timestamp = df.loc[:,'timestamp'].values #timestamp value
np_datetime = df.loc[:,'dttm_utc'].values #yyyy-mm-dd and 24-hour time
np_powervalue = df.loc[:,'value'].values #power reading of site in kWh
np_estimated = df.loc[:,'estimated'].values #boolean (0 | 1) if reading
was estimated
np_anomaly = df.loc[:,'anomaly'].values #non-blank value if reading was
erroneous

# Chart for load on weekdays vs weekends
df_datetime = pd.to_datetime(df['dttm_utc'])
df_days_of_week = df_datetime.dt.weekday #0-6 for Mon-Sun
df_month = df_datetime.dt.month # returns month of value January (1) -
December (12)
np_days_of_week = df_days_of_week.values #convert to np_array
np_month = df_month.values #convert to np_array

np_load_weekends = np.array([]) #Sat(5) & Sun(6)
np_load_weekdays = np.array([]) #Mon(0), Tues(1), Wed(2), Thu(3), Fri(4)

idx = 0 #index var

#Get load on weekends and weekdays
for day in np_days_of_week:
    if day > 4:
        np_load_weekends = np.append(np_load_weekends, np_powervalue[idx])
        np_load_weekdays = np.append(np_load_weekdays, np.nan) #equalize
values for graph
    else:
        np_load_weekdays = np.append(np_load_weekdays, np_powervalue[idx])
        np_load_weekends = np.append(np_load_weekends, np.nan)
    idx = idx + 1
```

```
#Display x-axis ticks in values of first of each month (1-12)
ir = 1 #ignore first value of array (first of month)
while ir < np_month.size:
    comparison_val = 0
    if ir+1 < np_month.size: #prevent out-of-bounds
        comparison_val = np_month[ir+1]
    if np_month[ir] == comparison_val:
        np_month[ir] = 0 #set to zero value (null representation)
    elif np_month[ir] != comparison_val: #case where transition between
month
        np_month[ir] = 0
        ir += 1 #skip first of each month
    ir += 1
np_month[np_month.size - 1] = 0 #ignore last value (month turnover to next
year)

x_t = range(len(np_month))
plt.plot(x_t, np_load_weekdays, label="Power Load of Site during
Weekdays")
plt.plot(x_t, np_load_weekends, label="Power Load of Site during
Weekends")
plt.title("Site 690 Power Load Weekdays vs Weekends")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Months of the Year (2012-2013)")
plt.legend()
plt.xticks(x_t, np_month)

#improve appearance of x-axis labels
xt2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xt2)):
    if np_month[i] == 0:
        xt2[i].set_visible(False)

plt.show()

#General load graph of all data points throughout the year
maxload = np.amax(np_powervalue)
minload = np.amin(np_powervalue)
max_load_idx = np.where(np_powervalue == maxload) #get index of max load
in array
max_load_idx = int(max_load_idx[0])
min_load_idx = np.where(np_powervalue == minload) #get index of min load
in array
min_load_idx = min_load_idx[0] #particular to my data: convert from tuple
to array
min_load_idx = min_load_idx[0] #particular to my data: convert from array
```

```
to scalar
max_load_formatted = "{:.2f}".format(maxload)
min_load_formatted = "{:.2f}".format(minload)
max_load_label = "Max Load = " + max_load_formatted + " (kW/h)"
min_load_label = "Min Load = " + min_load_formatted + " (kW/h)"

x_t = range(len(np_month))
plt.plot(x_t, np_powervalue, label="Power Load of Site Every Day of Year")
plt.annotate(max_load_label, (max_load_idx, maxload), textcoords="offset
points", xytext=(0, 5), ha="center")
plt.annotate(min_load_label, (min_load_idx, minload), textcoords="offset
points", xytext=(0, -5), ha="center")
plt.title("Site 690 Power Load through 2012-2013")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Months of the Year (2012-2013)")
plt.legend()
plt.xticks(x_t, np_month)

#improve appearance of x-axis labels
xt2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xt2)):
    if np_month[i] == 0:
        xt2[i].set_visible(False)

plt.show()


#Very ugly and large algorithm to calculate CBL (Customer Baseline Load)
... but it works
def CBL_calculator(eligible_days, dr_event_times):
    #eligible_days is an array of strings: 10 eligible days for CBL
calculation in yyyy-mm-dd format
    #dr_event_times is an array of ints: these are the hours in 24 hr time
that the dr event occurred
    days_powervalues = np.zeros(shape=(10, 288)) #10 rows, 288 columns
    avg_event_period_usage = np.zeros(10) #kW/h
    total_event_period_usage = np.zeros(10) #kW
    five_highest_days_indices = np.zeros(5) #holds the indices of the 5
highest usage days in days_powervalues
    # [highest day index, ... , fifth highest day index]
    avg_day_cbl = np.zeros(288) #final result
    iter1 = 0
    for day in eligible_days:
        iter2 = 0
        iter3 = 0
        #get values for each day
        for val in np_datetime:
            if(val.startswith(day)):
                days_powervalues[iter1][iter2] = np_powervalue[iter3]
                iter2 = iter2 + 1
            iter3 = iter3 + 1
        #calculate avg_event_period_usage and total_event_period_usage for
dr event times
        total_total_usage = 0
        total_total_cnt = 0
        for hr in dr_event_times:
            total_usage = 0
            total_cnt = 0
            five_min_counter = 0
            for i in range(288): #go through each col of matrix (whole
day)
                if(floor(five_min_counter / 60) == hr):
                    total_usage += days_powervalues[iter1][i]
                    total_cnt += 1
                five_min_counter = five_min_counter + 5

            total_total_usage += total_usage
            total_total_cnt += total_cnt
        avg_event_day_usage = total_total_usage / total_total_cnt
        total_event_day_usage = avg_event_day_usage * len(dr_event_times)
        avg_event_period_usage[iter1] = avg_event_day_usage
        total_event_period_usage[iter1] = total_event_day_usage

        iter1 = iter1 + 1
    #get indices of 5 highest usage days
    for i in range(5):
        max = total_event_period_usage[i]
        max_idx = 0
        for j in range(10):
            if(total_event_period_usage[j] > max and i != j):
                max = total_event_period_usage[j]
                max_idx = j
        total_event_period_usage[max_idx] = 0
        five_highest_days_indices[i] = max_idx

    five_highest_days_indices = five_highest_days_indices.astype(int)
    #calculate final result
    for x in range(avg_day_cbl.size): #for each column
        total = 0
        for y in range(five_highest_days_indices.size): #for each selected
row
            total += days_powervalues[five_highest_days_indices[y]][x]
        avg_day_cbl[x] = total / 5
    return avg_day_cbl


#DR event (6/21/2012) load graph and baseline (CBL)
# days should be in descending order
days = ["2012-06-18", "2012-06-15", "2012-06-14", "2012-06-13",
"2012-06-12",
        "2012-06-11", "2012-06-08", "2012-06-07", "2012-06-06",
"2012-06-05"]
event_times = [13, 14, 15, 16, 17] #DR event from 13:00 to 18:00
dr_event_day_cbl = CBL_calculator(days, event_times)
np_dr_event_day_powervalues = np.array([])
np_dr_event_times = np.array([])

#Get actual power values from day of dr event and hr values (for x-axis
tick labels)
idx2 = 0
dr_day = "2012-06-21"
for val in np_datetime:
    if(np_datetime[idx2].startswith(dr_day)):
        np_dr_event_day_powervalues =
np.append(np_dr_event_day_powervalues, np_powervalue[idx2])
        temp_str = val.split()
        temp_str = temp_str[1]
        #truncate string to remove seconds
        np_dr_event_times = np.append(np_dr_event_times, temp_str[0:5])
    idx2 = idx2 + 1

x_ticks = range(len(np_dr_event_times))
plt.plot(x_ticks, np_dr_event_day_powervalues, label="Power Load of Site
during DR Day")
plt.plot(x_ticks, dr_event_day_cbl, label="Customer Baseline Load (CBL)
during DR Day")
plt.title("Site 690 Power Load during DR Event: June 21st, 2012")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Time")
plt.xticks(x_ticks, np_dr_event_times, rotation=45) #set x-axis values to
time intervals

#improve appearance of x-axis labels
xticks2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xticks2)):
    if i % 12 != 0:
        xticks2[i].set_visible(False)

#Add highlighted x-span to show DR event activation
plt.axvspan(144, 216, color='red', alpha=0.25, label="DR Event
Activation")
plt.legend()
plt.show()

#Min and max load of site
max_load = np.amax(np_powervalue)
max_load_str = "Max Load of Site = " + str(max_load) + " kW/h"
print(max_load_str)
min_load = np.amin(np_powervalue)
min_load_str = "Min Load of Site = " + str(min_load) + " kW/h"
print(min_load_str)

#Percentage of missing values (the error values)
num_rows_df_og = df_og.shape[0]
num_rows_df = df.shape[0]
percentage = (1 - (num_rows_df / num_rows_df_og)) * 100
percent_formatted = "{:.4f}".format(percentage)
percentage_str = "Percentage of missing data points = " +
percent_formatted + "%"
print(percentage_str)
```

Appendix B

Site 697 Code: data analysis and plotting

```
from math import floor
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df_og = pd.read_csv(r"C:\Users\dylan\Desktop\ELEN285 project\csv\697.csv")
#unmodified dataframe
df = df_og.drop(df_og[df_og['value'] == 0].index) #remove all rows with
error values (0 kWh)

#Numpy arrays of each column in the csv
np_timestamp = df.loc[:,'timestamp'].values #timestamp value
np_datetime = df.loc[:,'dttm_utc'].values #yyyy-mm-dd and 24-hour time
np_powervalue = df.loc[:,'value'].values #power reading of site in kWh
np_estimated = df.loc[:,'estimated'].values #boolean (0 | 1) if reading
was estimated
np_anomaly = df.loc[:,'anomaly'].values #non-blank value if reading was
erroneous

# Chart for load on weekdays vs weekends
df_datetime = pd.to_datetime(df['dttm_utc'])
```

```
df_days_of_week = df_datetime.dt.weekday #0-6 for Mon-Sun
df_month = df_datetime.dt.month # returns month of value January (1) -
December (12)
np_days_of_week = df_days_of_week.values #convert to np_array
np_month = df_month.values #convert to np_array

np_load_weekends = np.array([]) #Sat(5) & Sun(6)
np_load_weekdays = np.array([]) #Mon(0), Tues(1), Wed(2), Thu(3), Fri(4)

idx = 0 #index var

#Get load on weekends and weekdays
for day in np_days_of_week:
    if day > 4:
        np_load_weekends = np.append(np_load_weekends, np_powervalue[idx])
        np_load_weekdays = np.append(np_load_weekdays, np.nan) #equalize
values for graph
    else:
        np_load_weekdays = np.append(np_load_weekdays, np_powervalue[idx])
        np_load_weekends = np.append(np_load_weekends, np.nan)
    idx = idx + 1

#Display x-axis ticks in values of first of each month (1-12)
ir = 1 #ignore first value of array (first of month)
while ir < np_month.size:
    comparison_val = 0
    if ir+1 < np_month.size: #prevent out-of-bounds
        comparison_val = np_month[ir+1]
    if np_month[ir] == comparison_val:
        np_month[ir] = 0 #set to zero value (null representation)
    elif np_month[ir] != comparison_val: #case where transition between
month
        np_month[ir] = 0
        ir += 1 #skip first of each month
    ir += 1
np_month[np_month.size - 1] = 0 #ignore last value (month turnover to next
year)

x_t = range(len(np_month))
plt.plot(x_t, np_load_weekdays, label="Power Load of Site during
Weekdays")
plt.plot(x_t, np_load_weekends, label="Power Load of Site during
Weekends")
plt.title("Site 697 Power Load Weekdays vs Weekends")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Months of the Year (2012-2013)")
plt.legend()
plt.xticks(x_t, np_month)

#improve appearance of x-axis labels
xt2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xt2)):
    if np_month[i] == 0:
        xt2[i].set_visible(False)

plt.show()

#General load graph of all data points throughout the year
maxload = np.amax(np_powervalue)
minload = np.amin(np_powervalue)
max_load_idx = np.where(np_powervalue == maxload) #get index of max load
in array
max_load_idx = int(max_load_idx[0])
min_load_idx = np.where(np_powervalue == minload) #get index of min load
in array
min_load_idx = min_load_idx[0] #particular to my data: convert from tuple
to array
min_load_idx = min_load_idx[0] #particular to my data: convert from array
to scalar
max_load_formatted = "{:.2f}".format(maxload)
min_load_formatted = "{:.2f}".format(minload)
max_load_label = "Max Load = " + max_load_formatted + " (kW/h)"
min_load_label = "Min Load = " + min_load_formatted + " (kW/h)"

x_t = range(len(np_month))
plt.plot(x_t, np_powervalue, label="Power Load of Site Every Day of Year")
plt.annotate(max_load_label, (max_load_idx, maxload), textcoords="offset
points", xytext=(0, 5), ha="center")
plt.annotate(min_load_label, (min_load_idx, minload), textcoords="offset
points", xytext=(0, -5), ha="center")
plt.title("Site 697 Power Load through 2012-2013")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Months of the Year (2012-2013)")
plt.legend()
plt.xticks(x_t, np_month)

#improve appearance of x-axis labels
xt2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xt2)):
    if np_month[i] == 0:
        xt2[i].set_visible(False)

plt.show()
```

```
#Very ugly and large algorithm to calculate CBL (Customer Baseline Load)
... but it works
def CBL_calculator(eligible_days, dr_event_times):
    #eligible_days is an array of strings: 10 eligible days for CBL
calculation in yyyy-mm-dd format
    #dr_event_times is an array of ints: these are the hours in 24 hr time
that the dr event occurred
    days_powervalues = np.zeros(shape=(10, 288)) #10 rows, 288 columns
    avg_event_period_usage = np.zeros(10) #kW/h
    total_event_period_usage = np.zeros(10) #kW
    three_highest_days_indices = np.zeros(3) #holds the indices of the 5
highest usage days in days_powervalues
    # [highest day index, ... , fifth highest day index]
    avg_day_cbl = np.zeros(288) #final result
    iter1 = 0
    for day in eligible_days:
        iter2 = 0
        iter3 = 0
        #get values for each day
        for val in np_datetime:
            if(val.startswith(day)):
                days_powervalues[iter1][iter2] = np_powervalue[iter3]
                iter2 = iter2 + 1
            iter3 = iter3 + 1
        #calculate avg_event_period_usage and total_event_period_usage for
dr event times
        total_total_usage = 0
        total_total_cnt = 0
        for hr in dr_event_times:
            total_usage = 0
            total_cnt = 0
            five_min_counter = 0
            for i in range(288): #go through each col of matrix (whole
day)
                if(floor(five_min_counter / 60) == hr):
                    total_usage += days_powervalues[iter1][i]
                    total_cnt += 1
                five_min_counter = five_min_counter + 5

            total_total_usage += total_usage
            total_total_cnt += total_cnt
        avg_event_day_usage = total_total_usage / total_total_cnt
        total_event_day_usage = avg_event_day_usage * len(dr_event_times)
        avg_event_period_usage[iter1] = avg_event_day_usage
        total_event_period_usage[iter1] = total_event_day_usage
        iter1 = iter1 + 1
    #get indices of 5 highest usage days
    for i in range(3):
        max = total_event_period_usage[i]
        max_idx = 0
        for j in range(10):
            if(total_event_period_usage[j] > max and i != j):
                max = total_event_period_usage[j]
                max_idx = j
        total_event_period_usage[max_idx] = 0
        three_highest_days_indices[i] = max_idx

    three_highest_days_indices = three_highest_days_indices.astype(int)
    #calculate final result
    for x in range(avg_day_cbl.size): #for each column
        total = 0
        for y in range(three_highest_days_indices.size): #for each
selected row
            total += days_powervalues[three_highest_days_indices[y]][x]
        avg_day_cbl[x] = total / 3

    return avg_day_cbl


#DR event (6/21/2012) load graph and baseline (CBL)
# days should be in descending order
days = ["2012-08-12", "2012-08-11", "2012-08-10", "2012-08-09",
"2012-08-08",
        "2012-08-07", "2012-08-06", "2012-08-05", "2012-08-04",
"2012-08-03"]
event_times = [16, 17] #DR event from 12:00 to 18:00
dr_event_day_cbl = CBL_calculator(days, event_times)
np_dr_event_day_powervalues = np.array([])
np_dr_event_times = np.array([])

#Get actual power values from day of dr event and hr values (for x-axis
tick labels)
idx2 = 0
dr_day = "2012-08-13"
for val in np_datetime:
    if(np_datetime[idx2].startswith(dr_day)):
        np_dr_event_day_powervalues =
np.append(np_dr_event_day_powervalues, np_powervalue[idx2])
        temp_str = val.split()
        temp_str = temp_str[1]
        #truncate string to remove seconds
        np_dr_event_times = np.append(np_dr_event_times, temp_str[0:5])
    idx2 = idx2 + 1

x_ticks = range(len(np_dr_event_times))
plt.plot(x_ticks, np_dr_event_day_powervalues, label="Power Load of Site
during DR Day")
plt.plot(x_ticks, dr_event_day_cbl, label="Customer Baseline Load (CBL)
during DR Day")
plt.title("Site 697 Power Load during DR Event: August 13, 2012")
plt.ylabel("Power Reading (kW/h)")
plt.xlabel("Time")
plt.xticks(x_ticks, np_dr_event_times, rotation=45) #set x-axis values to
time intervals

#improve appearance of x-axis labels
xticks2 = plt.gca().xaxis.get_major_ticks()
for i in range(len(xticks2)):
    if i % 12 != 0:
        xticks2[i].set_visible(False)

#Add highlighted x-span to show DR event activation
plt.axvspan(192, 204, color='red', alpha=0.25, label="DR Event
Activation")
plt.legend()
plt.show()
#Min and max load of site
max_load = np.amax(np_powervalue)
max_load_str = "Max Load of Site = " + str(max_load) + " kW/h"
print(max_load_str)
min_load = np.amin(np_powervalue)
min_load_str = "Min Load of Site = " + str(min_load) + " kW/h"
print(min_load_str)
#Percentage of missing values (the error values)
num_rows_df_og = df_og.shape[0]
num_rows_df = df.shape[0]
percentage = (1 - (num_rows_df / num_rows_df_og)) * 100
percent_formatted = "{:.4f}".format(percentage)
percentage_str = "Percentage of missing data points = " +
percent_formatted + "%"
print(percentage_str)
```