User Guide: OpenEIS Reference Code

Jessica Granderson
David Lorenzetti
Claudine Custodio

Lawrence Berkeley National Laboratory

Srinivas Katipamula
Hung Ngo
Robert Lutes

Pacific Northwest National Laboratory

Prepared for:
George Hernandez
DOE Building Technologies Office
February 10, 2014

Table of Contents

Purpose

This User Guide serves three key purposes:

1. It describes the algorithms whose source code and pseudo code is made publically available through the OpenEIS project.

2. It describes how these algorithms can be used by individuals who wish to directly apply the code to analyze building data.

3. It describes how the source code and pseudo code can be used as a *reference implementation* by developers and programmers who wish to adapt these algorithms for use in commercial tools or service offerings. Commercial implementations may incorporate diverse and more sophisticated interfaces, user options, and visual representations. These options are noted in the description of each algorithm.

Algorithms Overview

This section presents overviews of each algorithm included in the collection of Open Energy Information System (OpenEIS) code, summarizing how each is used to gain insights about building energy performance, operations, and comfort.

Time Series Load Profiling

Time series load profiling is used on a daily or weekly basis to understand the relationship between energy use and time of day. Abnormalities or changes in load profiles can indicate inefficiencies due to scheduling errors, unexpected or irregular equipment operation, high use during unoccupied hours, or untimely peaks.

Plots of at least 24-hour periods of interval meter data ("profiles") are inspected and evaluated in the context of the building's operational hours and intended system control schedules. Changes in load size and shape against time of day, day of week, or season are considered. Unexplainable differences may indicate operating errors or equipment faults, and therefore energy waste, and should be investigated.

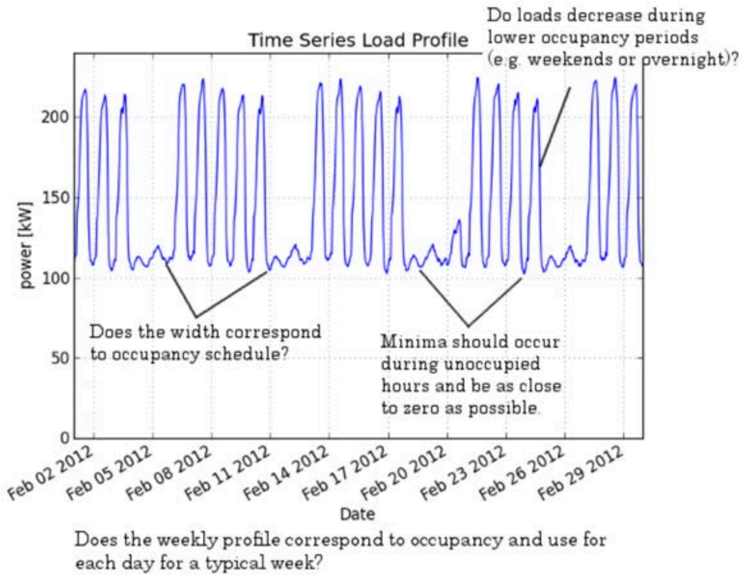


Figure 1. OpenEIS reference code output for time series load profiling

Time series load profiling is reviewed in detail in pages 71–79 in the *Energy Information Handbook*. In the OpenEIS reference code, kilowatts (kW) are plotted on the y-axis, with time on the x-axis. The plot is limited the most recent month of data in the dataset. More flexible implementations might allow options such as auto-scaling based on the amount of data, user-definable subsets of data, adjustable x- and y-axes with different zoom factors, "calendar" views that present the time series in rows and columns that correspond to weeks and days of the week, and more.

Heat Maps

Heat maps are a means of visualizing and presenting the information that is contained in a time series load profile. The maps color-code the size of the load so that "hot spots" and patterns are easily identified. Time of day is plotted on the x-axis, and day or date is indicated on the y-axis (or vice versa). The heat map is inspected and evaluated in the context of the building's operational hours and schedules, or intended system control schedules. Depending on the historic length of data that is plotted (daily, weekly, or seasonal scheduling), peak and start-up/shut down opportunities are revealed.

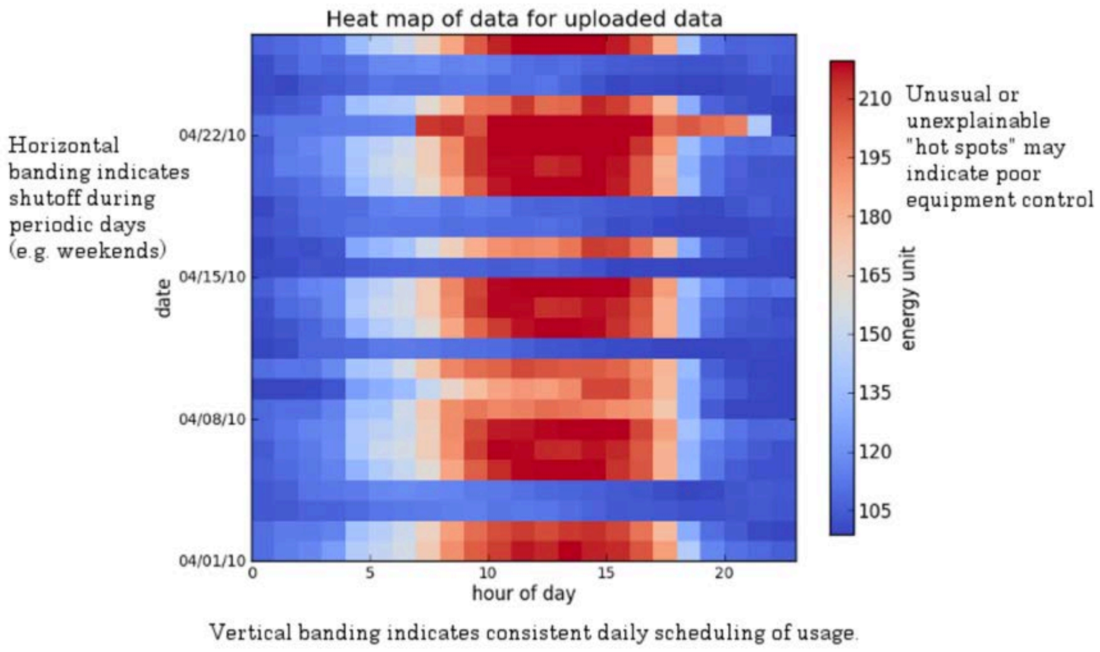


Figure 2. OpenEIS reference code output for heat maps

Heat maps are reviewed in pages 242–243 in the *Energy Information Handbook*. In the OpenEIS reference code, blue coloring corresponds to low or minimum building load, and red coloring corresponds to high or maximum building loads. Loads are presented in units of kW, and hours of the day are plotted on the x-axis, and specific days on the y-axis. The plot is limited to the most recent year of data in the dataset. More flexible implementations might allow options such as auto-scaling based on the total amount of data, plotting of user-defined subsets of data, or scaling the color bar from zero to the maximum observed load.

Energy Signature

Energy signatures are used to monitor and maintain the performance of temperature- dependent loads such as whole-building electric or gas use, or heating and cooling systems or components. They can reveal problems with insulation, outside air intake, or system efficiency.

Energy use for a given time interval is plotted against the corresponding average outdoor temperature in that period. Orderly data points reflect consistent behavior, while highly scattered data points indicate potential inefficiency or lack of weather sensitivity. Other useful areas to examine are "base loads" (at which the energy use does not change with temperature) and the rate at which load changes with outside air temperature, known as the "heating slope" and/or "cooling slope."

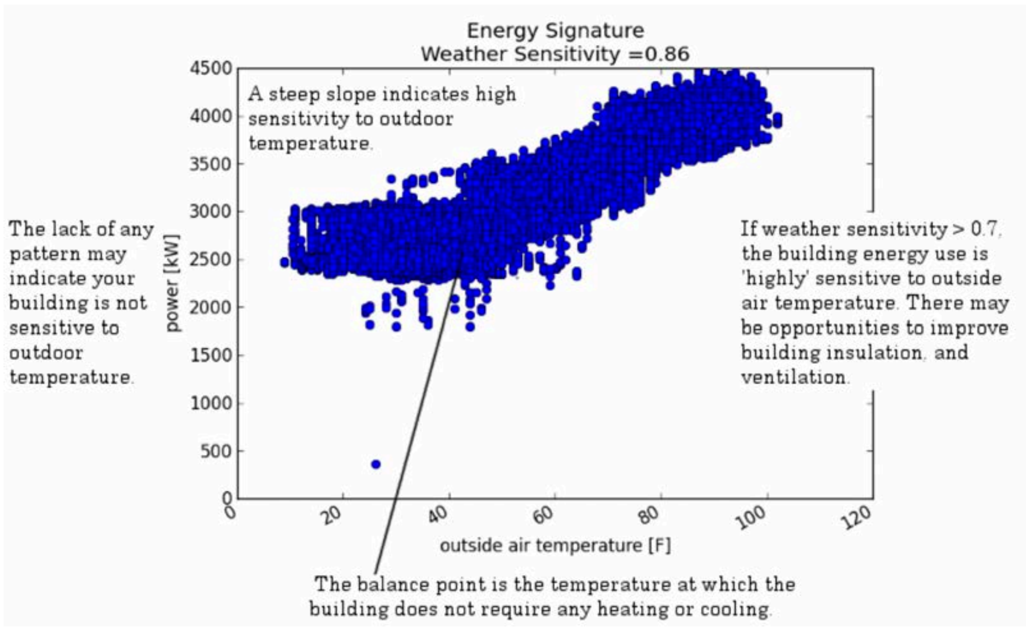


Figure 3. OpenEIS reference code output for energy signature

Energy signatures are reviewed in detail in pages 147–155 in the *Energy Information Handbook*. In the OpenEIS reference code, the energy metric is electric kW, and the temperature metric is degrees Fahrenheit. The plot is limited to the most recent year of data in the dataset. More sophisticated implementations might fit line segments to the data, making explicit note of the heating and cooling slopes and balance point. Enhanced flexibility would accommodate filtering to user-defined subsets of data, for example.

Weather Sensitivity

Weather sensitivity can be characterized by the Spearman rank-order correlation between building load and outside air temperature. This metric ranges from -1 to 1; however, for buildings, negative values are not expected. For example, if the value of the weather sensitivity metric is greater than 0.7, the building energy use is “highly” sensitive to outside air temperature, and there may be insulation, ventilation, or efficiency improvement opportunities.

In the OpenEIS reference code, the weather sensitivity metric is displayed as an overlay to the energy signature plot—a single summary statistic that contextualizes the shape of the energy signature. The metric is computed for the most recent year of data in the data set.

In the idealized case where there are no duplicate load values and no duplicate outside air temperatures, the Spearman rank-order correlation can be defined according to the equation below, as:

$$r_s = 1 - \frac{6\left(\sum D^2\right)}{N\left(N^2 - 1\right)}$$

where D is the difference between each pair of ranks

However, since duplicate temperatures and loads are likely in any sufficiently large data set, the OpenEIS implementation uses the defining equation, which finds the correlation coefficient (i.e., the Pearson product-moment correlation) between the rank orders of the loads and temperatures. Repeated load values are assigned the average rank for all the loads with that value, and similarly for temperatures. In addition, the OpenEIS implementation excludes from the analysis any load-temperature pair for which either the load or temperature datum is missing.

A more flexible version of this algorithm would allow filtering to user-defined sub-sets of data, for example to find correlations only during certain hours, only during weekdays, and so on.

Longitudinal Benchmarking

Longitudinal benchmarking compares the energy usage in a fixed period for a building, system, or component to that in a comparable "baseline" or "base" period of the same length, to determine if performance has deteriorated or improved, to set goals for a building or system, or to monitor for unexpectedly high usage.

Energy use in the base (reference) period is expressed according to a metric of choice, such as thousand Btu per square foot (KBtu/sf), forming a "benchmark." Performance is then tracked relative to the base-period benchmark.

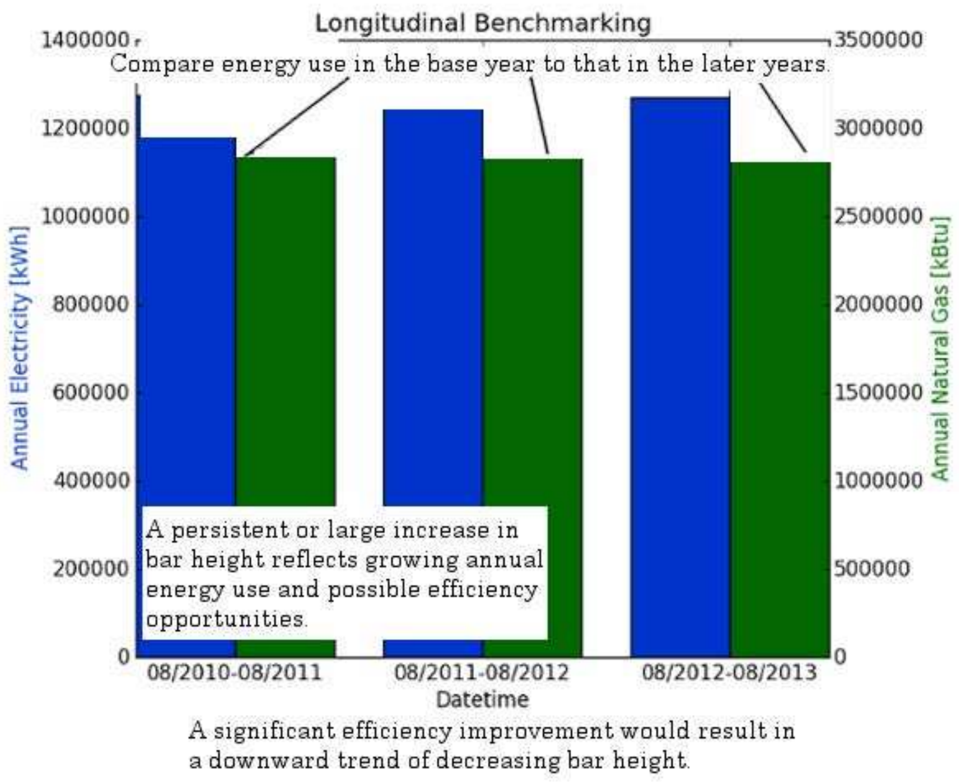


Figure 4. OpenEIS reference code output for longitudinal benchmarking

Longitudinal benchmarking is reviewed in detail in pages 51–58 in the *Energy Information Handbook*. In the OpenEIS reference code annual electricity and annual natural gas consumption is represented in kWh and kBtu, respectively, and plotted in vertical bars. The base year is defined as the first full year of data in the dataset. More flexible implementations would allow user-definable time periods other than annual usage, for example using a rolling 12-month totalization of energy use as opposed to a fixed base year.

Cross-Sectional Benchmarking

Cross-sectional benchmarking is used to compare a building's energy efficiency relative to a peer group. It is the first step in determining if performance is good or poor, and it shows how much potential there is to improve the building's efficiency.

Cross-sectional benchmarking results are usually expressed in terms of the percentile rank relative to the peer group. Fiftieth percentile means that half the buildings in the peer group are more efficient, and twenty-fifth percentile means that only 25 percent of the buildings are more efficient. Cross-sectional benchmarking is most often conducted at the whole-building level, commonly using an energy use intensity (EUI) metric such as combined fuels kBtu/sf.

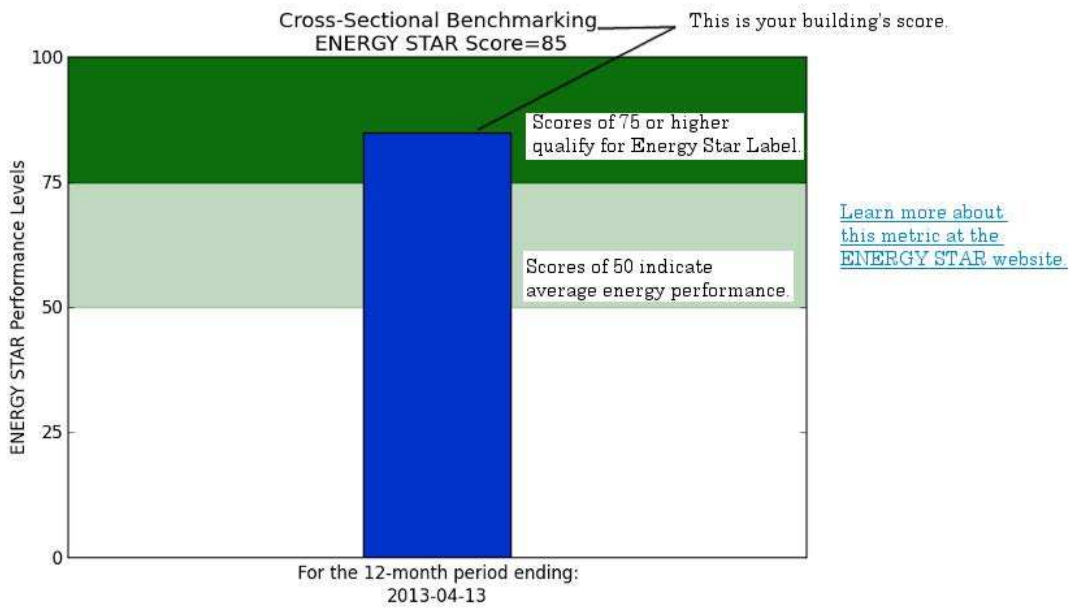
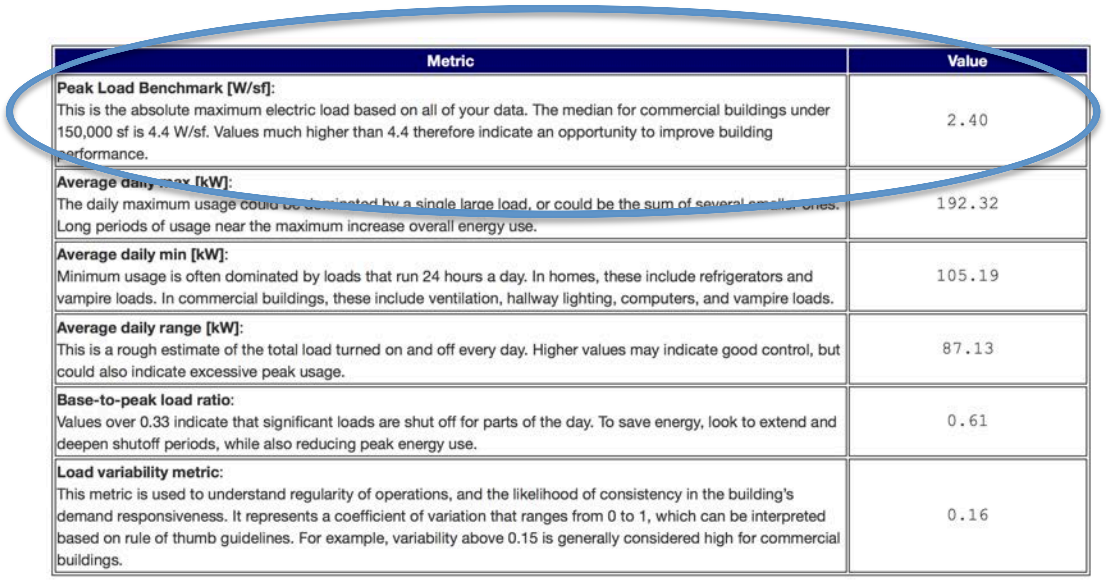


Figure 5. OpenEIS reference code output for cross-sectional benchmarking with formatting guidelines from ENERGY STAR.

Cross-sectional benchmarking is reviewed in detail in pages 59–67 in the *Energy Information Handbook*. In the OpenEIS reference code, the ENERGY STAR Target Finder web service is accessed to compute a benchmark score from 1 to 100, based on user-defined building characteristic inputs that account for factors such as building size, type, and climate. The ENERGY STAR score corresponds to the percentile rank, with higher values indicating higher efficiency. Scores of 75 or higher qualify for the ENERGY STAR label. The OpenEIS reference code uses the most recent year of data in the dataset, and requires a basic set of building characteristics – gross floor area, zipcode, building type, and year of construction. Implementations that leverage additional user inputs such as hours of operation and number of occupants are recommended; they will give a more accurate score that is more customized to the specific building being analyzed. A complete list of additional building characteristics that can be sued is documented in ENERGY STAR Portfolio Manager web services page.

Peak Load Benchmarking

Peak load benchmarking is used to compare a building's peak electric load to a peer group. High peak loads can have a significant impact on utility costs in cases where demand charges are assessed, and are also a critical contributor to electrical reliability during times of extreme demand on the grid.



Metric	Value
Peak Load Benchmark [W/sf]: This is the absolute maximum electric load based on all of your data. The median for commercial buildings under 150,000 sf is 4.4 W/sf. Values much higher than 4.4 therefore indicate an opportunity to improve building performance.	2.40
Average daily max [kW]: The daily maximum usage could be dominated by a single large load, or could be the sum of several smaller ones. Long periods of usage near the maximum increase overall energy use.	192.32
Average daily min [kW]: Minimum usage is often dominated by loads that run 24 hours a day. In homes, these include refrigerators and vampire loads. In commercial buildings, these include ventilation, hallway lighting, computers, and vampire loads.	105.19
Average daily range [kW]: This is a rough estimate of the total load turned on and off every day. Higher values may indicate good control, but could also indicate excessive peak usage.	87.13
Base-to-peak load ratio: Values over 0.33 indicate that significant loads are shut off for parts of the day. To save energy, look to extend and deepen shutoff periods, while also reducing peak energy use.	0.61
Load variability metric: This metric is used to understand regularity of operations, and the likelihood of consistency in the building's demand responsiveness. It represents a coefficient of variation that ranges from 0 to 1, which can be interpreted based on rule of thumb guidelines. For example, variability above 0.15 is generally considered high for commercial buildings.	0.16

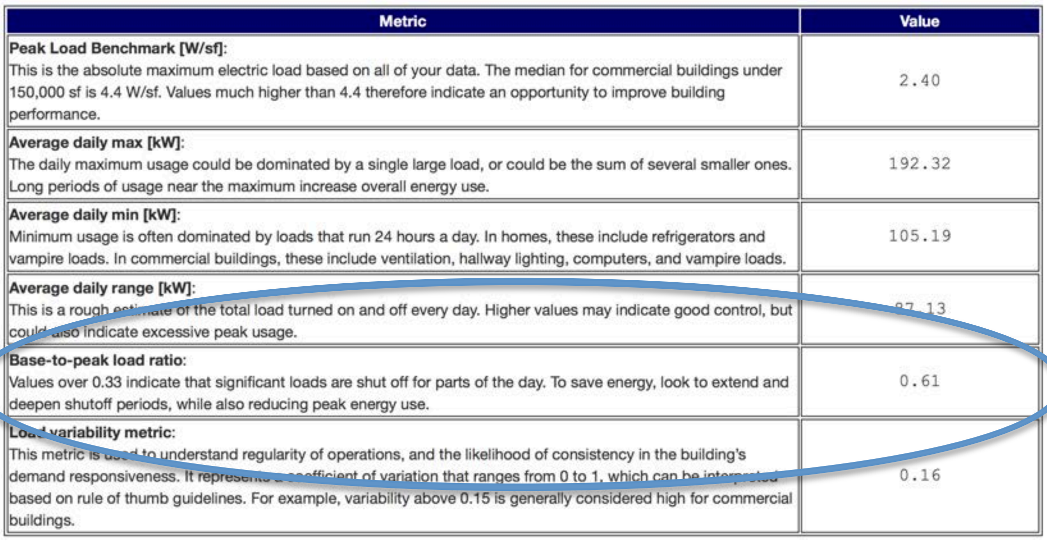
Figure 6. OpenEIS reference code output for peak load benchmarking

Peak load benchmarking is addressed in the Peak Load Analysis example on page 85 in the *Energy Information Handbook*. In the OpenEIS reference code, peak load is defined as "the absolute maximum load that appears in the data set, and is normalized by building area, for a watts per square foot (W/sf) metric." The EnergyIQ benchmarking tool was used to generate the median peak load against which a user's data can be compared. The California Commercial End-Use Survey data set was used, and the peer group was defined as: all commercial building types and vintages, from all California climates, with floor areas less than 150,000 sf. For this peer group, the median peak is 4.4 W/sf. In the OpenEIS reference code, peak load benchmarking results are presented in a summary table of key metrics; however, peak load benchmarking results may also be presented with a visual display of the building load profile or other developer-defined presentations.

Base-to-Peak Load Ratios

Base-to-peak load ratios compare the minimum building load to the maximum building load, to judge whether the building is "shut down" after hours. A ratio closer to zero indicates a large difference between the smallest and largest building loads, whereas a ratio closer to zero indicates a near-static, non-fluctuating load, and therefore an opportunity to improve efficiency. Improvements can be made by increasing the duration of scheduled equipment-off times and by increasing the number of loads that are shut off after hours.

A "good" versus "poor" value of base-to-peak load depends on the specific building operations and characteristics; however, ratios less than approximately 0.33 indicate that significant loads are shut off for parts of the day.



Metric	Value
Peak Load Benchmark [W/sf]: This is the absolute maximum electric load based on all of your data. The median for commercial buildings under 150,000 sf is 4.4 W/sf. Values much higher than 4.4 therefore indicate an opportunity to improve building performance.	2.40
Average daily max [kW]: The daily maximum usage could be dominated by a single large load, or could be the sum of several smaller ones. Long periods of usage near the maximum increase overall energy use.	192.32
Average daily min [kW]: Minimum usage is often dominated by loads that run 24 hours a day. In homes, these include refrigerators and vampire loads. In commercial buildings, these include ventilation, hallway lighting, computers, and vampire loads.	105.19
Average daily range [kW]: This is a rough estimate of the total load turned on and off every day. Higher values may indicate good control, but could also indicate excessive peak usage.	87.13
Base-to-peak load ratio: Values over 0.33 indicate that significant loads are shut off for parts of the day. To save energy, look to extend and deepen shutoff periods, while also reducing peak energy use.	0.61
Load variability metric: This metric is used to understand regularity of operations, and the likelihood of consistency in the building's demand responsiveness. It represents a coefficient of variation that ranges from 0 to 1, which can be interpreted based on rule of thumb guidelines. For example, variability above 0.15 is generally considered high for commercial buildings.	0.16

Figure 7. OpenEIS reference code output for base-to-peak load ratio

Base-to-peak load ratios are covered under the Peak Load Analysis examples in pages 86–87 in the *Energy Information Handbook*. In the OpenEIS reference code, the base load is defined as the 5th percentile, and the peak load is defined as the 95th percentile. The *absolute* maximum and minimum load are purposely excluded, to avoid one-off cases. The base and peak loads are computed for each day in the data set. The base-to-peak load is calculated for each day, and the average of these ratios is the average daily base-to-peak load ratio. In the reference code, the ratio is presented in a summary table of key metrics; however, analysis of peak-to-base loads may also be presented with a visual display of the building load profile. More sophisticated implementations might filter weeks and holidays, or accommodate other user-defined filtering options.

Load Duration Curve

Load duration curves are used to understand the number of hours or percentage of time during which the building load is at or below a certain value. Ideally, the highest loads should occur for a smaller fraction of the time. If the building is near its peak load for a significant portion of the time, the HVAC equipment could be undersized, affecting comfort; conversely, there may be systems that are running more continuously than necessary. If the load is near peak for only a short duration of time, there may be an opportunity to reduce peak demand charges.

In a load duration curve, the y-axis indicates the building load, and the x-axis indicates the percent of the time (or total number of hours) that the load is at or below that load. A curve that is steep on the left side of the plot indicates that high loads are present a small fraction of the time; whereas, as a curve that is steep on the right side indicates the reverse.

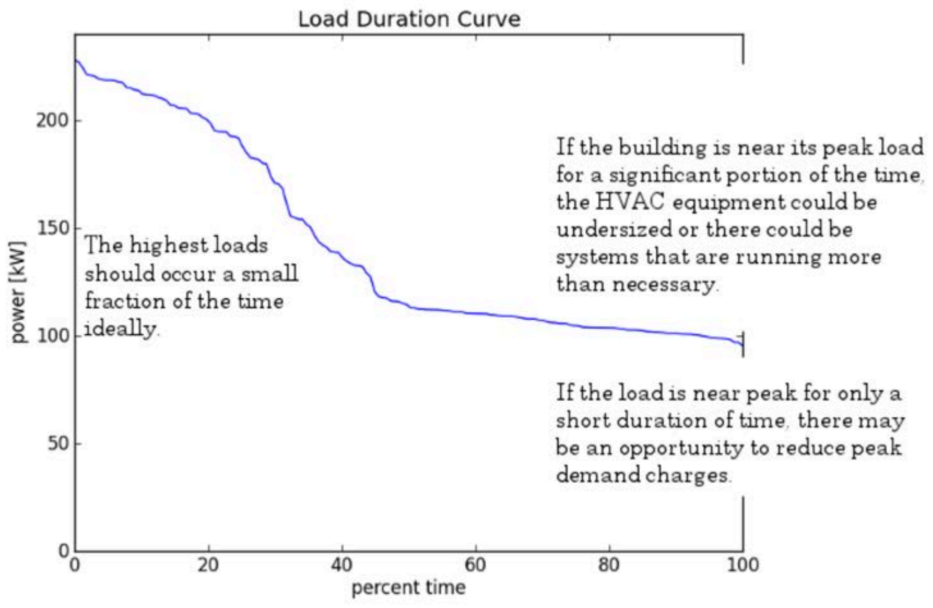
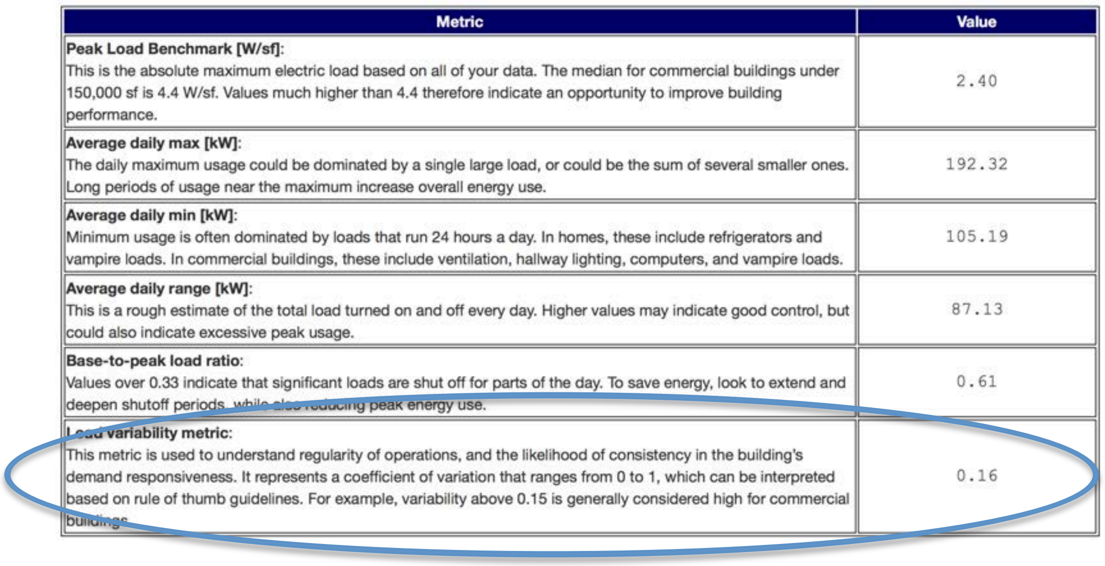


Figure 8. OpenEIS reference code output for load duration curve

Load duration curves are covered in the Peak Load Analysis example on page 89 in the *Energy Information Handbook*. In the OpenEIS reference code, the x-axis is the percentage of the total time that is spanned by the data; the y-axis ranges from zero to the maximum load that appears in the data set. The plot in the OpenEIS reference code is limited to the most recent year of data in the dataset.

Load Variability

Load variability is a measure of the degree to which whole-building loads are regular and predictable. It is a metric that is used to understand regularity of operations, and the likelihood of consistency in the building's demand responsiveness. It represents a coefficient of variation that ranges from 0 to 1, which can be interpreted based on rule-of-thumb guidelines. For example, variability above 0.15 is generally considered high for commercial buildings.



Metric	Value
Peak Load Benchmark [W/sf]: This is the absolute maximum electric load based on all of your data. The median for commercial buildings under 150,000 sf is 4.4 W/sf. Values much higher than 4.4 therefore indicate an opportunity to improve building performance.	2.40
Average daily max [kW]: The daily maximum usage could be dominated by a single large load, or could be the sum of several smaller ones. Long periods of usage near the maximum increase overall energy use.	192.32
Average daily min [kW]: Minimum usage is often dominated by loads that run 24 hours a day. In homes, these include refrigerators and vampire loads. In commercial buildings, these include ventilation, hallway lighting, computers, and vampire loads.	105.19
Average daily range [kW]: This is a rough estimate of the total load turned on and off every day. Higher values may indicate good control, but could also indicate excessive peak usage.	87.13
Base-to-peak load ratio: Values over 0.33 indicate that significant loads are shut off for parts of the day. To save energy, look to extend and deepen shutoff periods, while also reducing peak energy use.	0.61
Load variability metric: This metric is used to understand regularity of operations, and the likelihood of consistency in the building's demand responsiveness. It represents a coefficient of variation that ranges from 0 to 1, which can be interpreted based on rule of thumb guidelines. For example, variability above 0.15 is generally considered high for commercial buildings.	0.16

Figure 9. OpenEIS reference code output for load variability

Load variability is covered in the Load Profiling example on page 78 of the *Energy Information Handbook*. In the OpenEIS reference code, load variability is presented in a summary table of key metrics.

In the OpenEIS implementation, load variability is defined as "the average of the time-of-day load variabilities." To find the time-of-day variability, collect all observations for a particular hour and find the variability for that time of day, as follows:

$$VAR = \frac{\sqrt{\dfrac{\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}{N-1}}}{\bar{x}}$$

where $\bar{x}$ is the average hourly load in the period,

and N is the number of days in the period

The load variability is the average of the 24 time-of-day variabilities calculated according to the equation above.

Whole-Building Energy Anomaly Detection

Whole-building energy anomaly detection is used to identify deviation of actual energy consumption from the expected consumption. Abnormal energy use can be isolated to a specific end-use or zone, with a combination of the user's knowledge of the building and supplementary data such as equipment schedules and outside air temperature. Anomaly detection is distinguished from simple alarming in that a baseline model is used to determine expected (typical) consumption. The baseline model accounts for key drivers of energy use such as outside air temperature and humidity, and hour of day and day of the week.

The output from the OpenEIS reference code is a plot of the daily 'Energy Consumption Index'. The index is calculated as a ratio of the actual daily energy consumption and the expected daily energy consumption. It indicates the degree of deviation of the actual energy consumption from the expected energy consumption. An index near one indicates the actual and expected energy consumption is nearly equal (normal operation). As the index approaches zero the expected consumption becomes increasingly larger than the actual consumption (actual less than expected). This means the building is using less energy than is typical. An index value of greater than one indicates the actual energy consumption is greater than the expected energy consumption. The building is using more energy than expected. For example, an index of 2.0 would indicate that the actual consumption is twice that of the expected consumption.

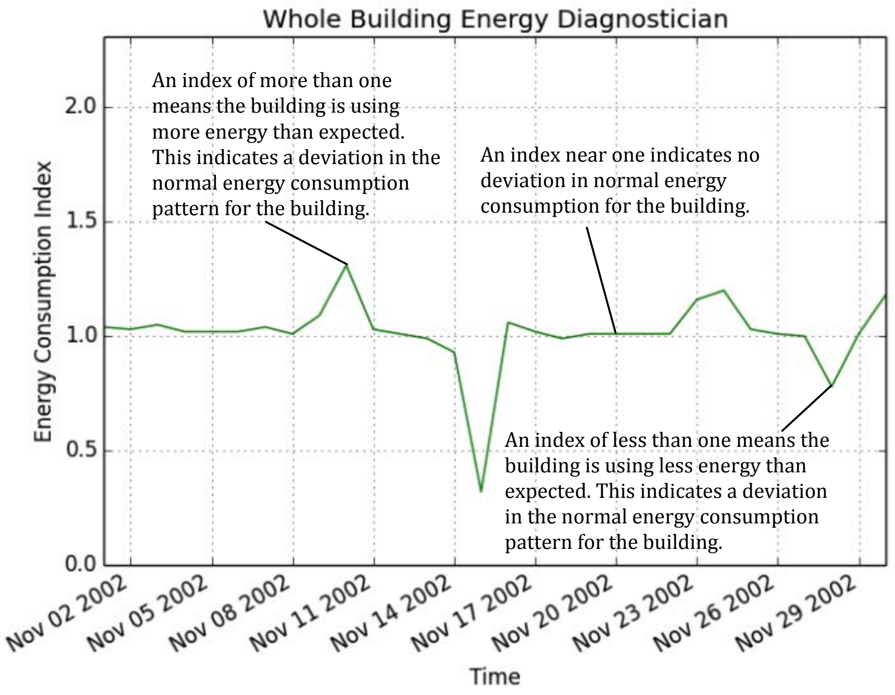


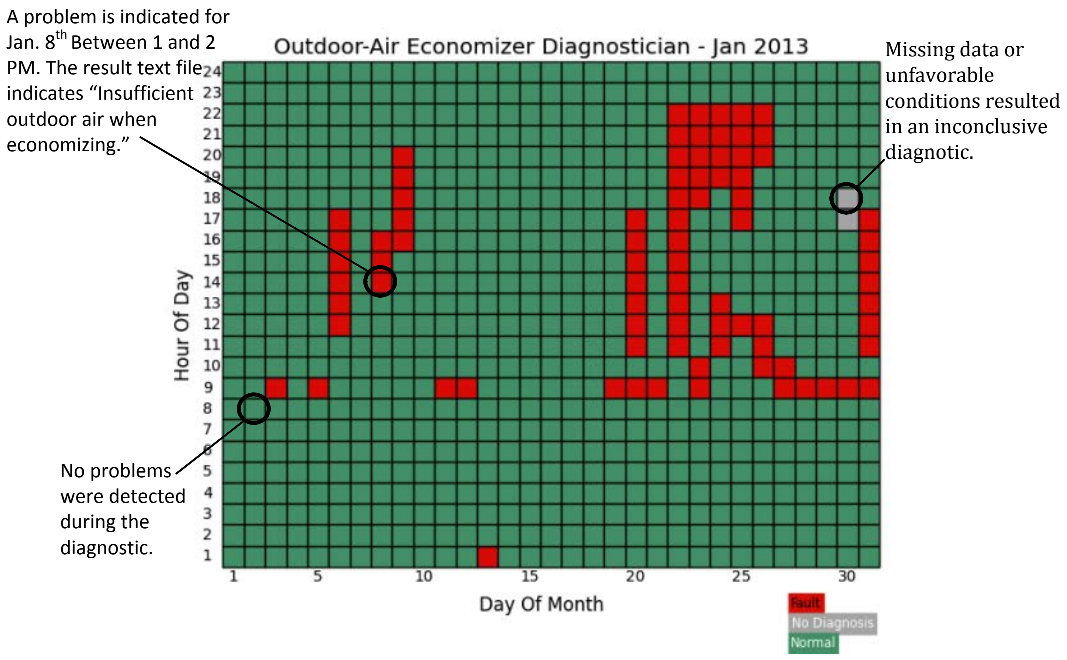
Figure 10. OpenEIS reference code output for whole-building energy anomaly detection

Energy anomaly detection is reviewed in detail on pages 175–182 in the *Energy Information Handbook*. In the OpenEIS reference code a “bin method” is used to create the baseline model. To develop an accurate baseline model, at least 9 months of data is recommended; therefore, this algorithm is best applied to data sets that contain at least 10 months of data. The configuration variables associated with the anomaly detection algorithm include: blended energy cost (energy and peak cost), identification of independent variables to be used in the baseline model, cost impact threshold, sensitivity setting, and the start and end dates for the baseline period.

Outside-Air Economizer Fault Detection and Diagnostics

Outside-air economizer (OAE) fault detection and diagnostics is used to identify problems in the operation and performance of air-handling units (AHUs) or packaged roof-top units (RTUs). Air-side economizers use free cooling by using cool outdoor air in place of (or to supplement) mechanical cooling when outdoor conditions are suitable for doing so. Unfortunately, economizers often do not work properly, causing energy-use penalties rather than savings. Common problems include incorrect control strategies, diverse types of damper linkage and actuator failures, and out-of-calibration sensors. These problems can be detected using sensor data used to control the system.

The OAE output is displayed by month with each column representing one day ascending from left to right and each square representing one hour ascending from bottom to top. Each hour can be assigned one of three colors: green when there were no problem detected, grey when the conditions were not favorable for the diagnostic for that hour, and red when there was a problem detected for that hour. The second image (Figure 12 – bottom) shows a time series plot with an estimation of the energy impact associated with the problems detected during the diagnostic period.



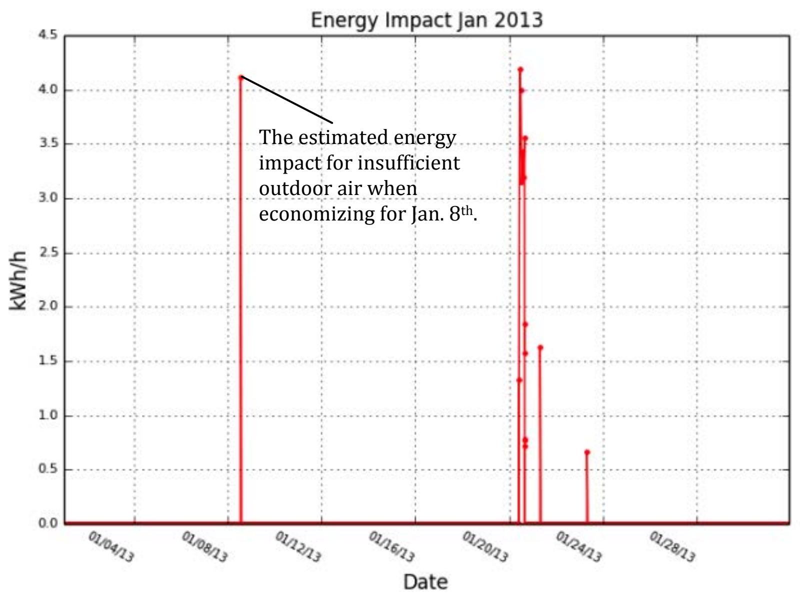


Figure 12. OpenEIS reference code output for outside-air economizer fault detection and diagnostics: Carpet plot to indicate the presence or absence of faults (top), and time series plot of the energy impacts associated with existing faults (bottom).

The OAE works best when using hourly data. When using sub-hourly data the OAE can aggregate the data. The data will be averaged over each hour of the day. If a user chooses to input sub-hourly data, and does not want to aggregate the data, the OAE will perform the diagnostic on each time entry (row) within the data input file. For example, if entering 15 minute interval data and choosing not to aggregate this data, the full set of OAE diagnostics will be performed on each 15 minute entry.

Fault detection and diagnostics is reviewed in detail on pages 183–196 in the *Energy Information Handbook;* specific outside air economizer diagnostics examples are included in pages 185–188, and 190–194. The OpenEIS reference code utilizes fault detection and diagnostic rules derived from engineering principles of proper and improper AHU/RTU operations.

A set of six checks are conducted to detect and diagnose faults in economizer and ventilation operations:

1. Identify measurement problems with outside air, mixed air and return air temperature sensors.
2. Determine whether the RTU/AHU is *not economizing* when it *should be*.
3. Determine whether the RTU/AHU *is economizing* when it should *not be*.
4. Determine whether the RTU/AHU is using excess outdoor air.
5. Determine whether the RTU/AHU is bringing in sufficient ventilation air.
6. Determine whether the RTU/AHU is operating outside the established operating schedule.

Data Requirements

This section summarizes the data requirements associated with each of the algorithms in the collection of OpenEIS reference code.

Table 1. Data Requirements for the OpenEIS Algorithms

Algorithm	Data Requirements	
	Data Type	**Minimum Resolution**
Time Series Load Profiling	Whole-building electric	Hourly
Heat Maps	Whole-building electric	
Energy Signature	Whole-building electric	
	Outside air temperature	
Weather Sensitivity	Whole-building electric	
	Outside air temperature	
Longitudinal Benchmarking	Whole-building electric	Annual
	Building floor area	n/a
Cross-Sectional Benchmarking	Whole-building electric	Annual
	Whole-building gas	
	Gross floor area; zipcode, building type; year of construction	n/a
Peak Load Benchmarking	Whole-building electric	Hourly
	Building floor area	n/a

Table 1. Data Requirements for the OpenEIS Algorithms (Continued)

Algorithm	Data Requirements	
	Data Type	**Minimum Resolution**
Base-to-Peak Load Ratios	Whole-building electric	Hourly
Load Duration Curves	Whole-building electric	
Load Variability	Whole-building electric	
Whole-Building Energy Anomaly Detection	Whole-building electric	
	Outside air temperature	
Outside-Air Economizer Fault Detection and Diagnostics	Outside air temperature	
	Return air, mixed air or discharge air temperature; outside air damper position signal; RTU/AHU schedule; heating and cooling command; fan status; cooling call	

Guidance for Direct Users

This section reviews the installation, computer hardware and software, and file formatting requirements that must be met to run the OpenEIS source code against a set of building energy data.

Data File Formatting Requirements

Data should be organized into the following files:

- A main data file, containing time-stamped outside air temperature, electricity, and gas data.
- Data files to support outside-air economizer diagnostics

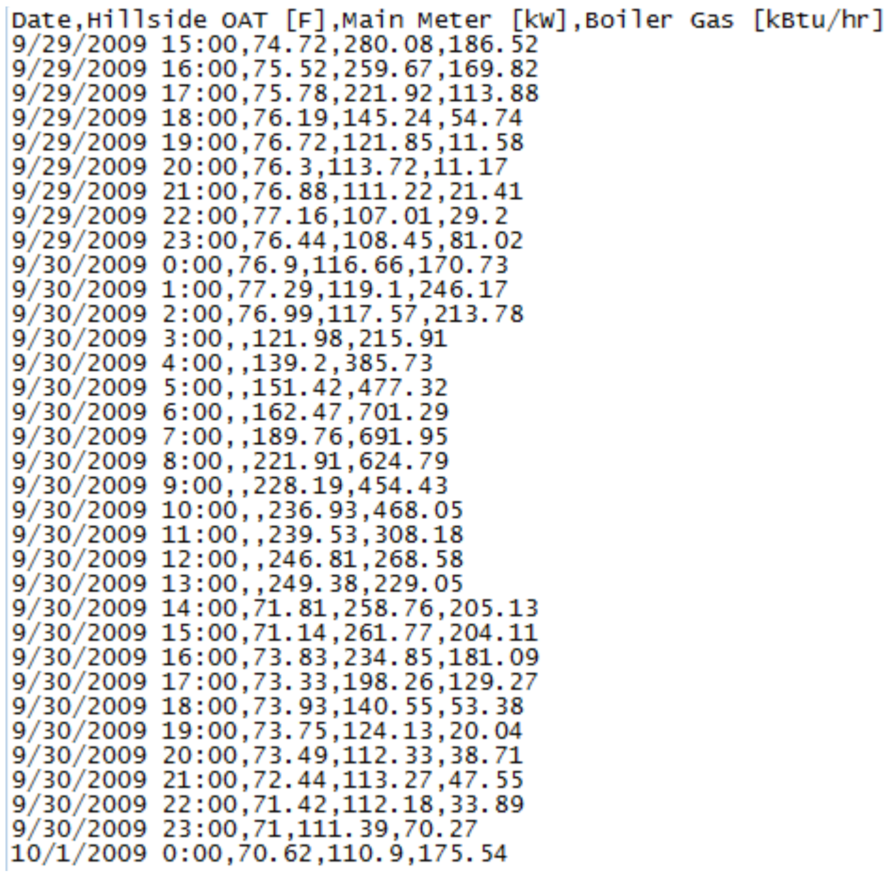
The main data file requirements are as follows:

- The file must have the following four columns, in this order: Date-Time, Outside Air Temperature, Main Electricity Meter, and Natural Gas Meter.
- Columns are comma-delimited.
- The first row is reserved for the column headers. The exact text of the column headers does not matter.
- Date-Times should be formatted as: (M)M/(D)D/YYYY (H)H:MM, with no quote marks or other punctuation.
- Outside Air Temperatures are floating-point numbers. Assumed to be [F].
- Main Electricity Meter data are floating-point numbers. Assumed to be [kW].
- Natural Gas Meter data are floating-point numbers. Assumed to be [kBtu/hr].

The data file requirements for outside air economizer diagnostics are as follows:

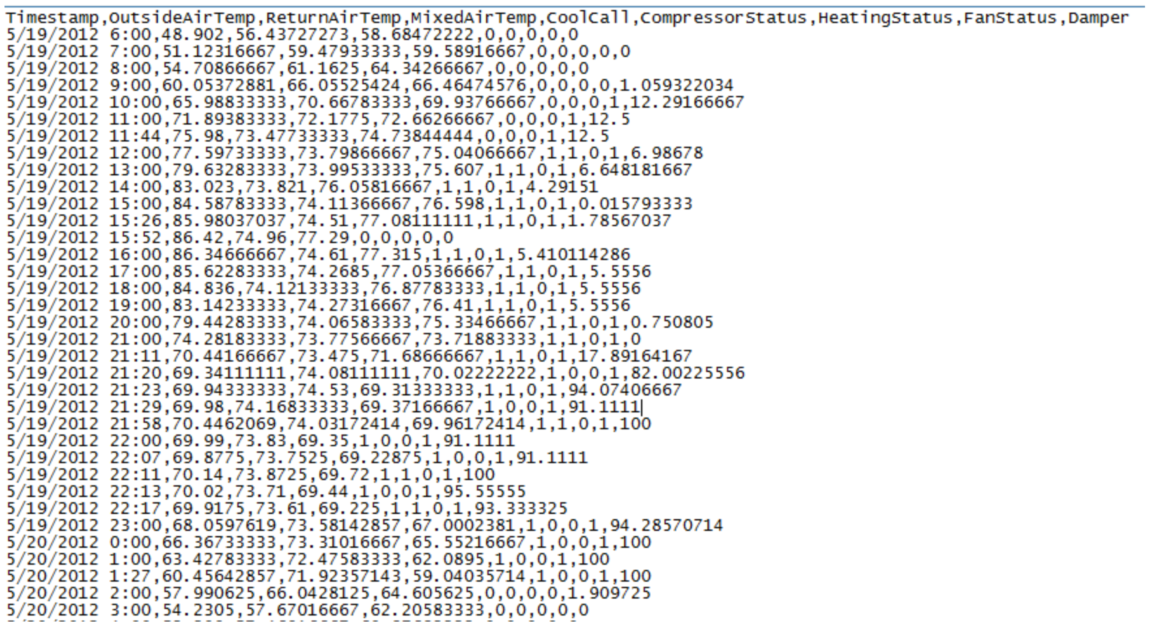
- The file must have the following columns, in this order: Date-time, outside air temperature, return air temperature, mixed air temperature or discharge air temperature, cooling call, cooling command, heating command, supply fan status, and outside air damper signal.
- Columns are comma-delimited.
- The first row is reserved for the column headers. The exact text of the column headers does not matter.
- Date-Times should be formatted as: (M)M/(D)D/YYYY (H)H:MM, with no quote marks or other punctuation.
- Outside air temperatures, return air temperatures, and mixed air temperatures are floating-point numbers. Assumed to be [F].
- Outside air damper signals are floating-point numbers. Assumed to be [%Open].
- Cooling call, cooling command, heating command, and supply fan status should be integers with '0' indicating 'OFF' and '1' indicating 'ON'. If the status is formatted as a floating-point number, the algorithm will internally reformat them as integers.

An example of correctly formatted main and outside air economizer data files are shown below in Figures 13 and 14.



```
Date,Hillside OAT [F],Main Meter [kW],Boiler Gas [kBtu/hr]
9/29/2009 15:00,74.72,280.08,186.52
9/29/2009 16:00,75.52,259.67,169.82
9/29/2009 17:00,75.78,221.92,113.88
9/29/2009 18:00,76.19,145.24,54.74
9/29/2009 19:00,76.72,121.85,11.58
9/29/2009 20:00,76.3,113.72,11.17
9/29/2009 21:00,76.88,111.22,21.41
9/29/2009 22:00,77.16,107.01,29.2
9/29/2009 23:00,76.44,108.45,81.02
9/30/2009 0:00,76.9,116.66,170.73
9/30/2009 1:00,77.29,119.1,246.17
9/30/2009 2:00,76.99,117.57,213.78
9/30/2009 3:00,,121.98,215.91
9/30/2009 4:00,,139.2,385.73
9/30/2009 5:00,,151.42,477.32
9/30/2009 6:00,,162.47,701.29
9/30/2009 7:00,,189.76,691.95
9/30/2009 8:00,,221.91,624.79
9/30/2009 9:00,,228.19,454.43
9/30/2009 10:00,,236.93,468.05
9/30/2009 11:00,,239.53,308.18
9/30/2009 12:00,,246.81,268.58
9/30/2009 13:00,,249.38,229.05
9/30/2009 14:00,71.81,258.76,205.13
9/30/2009 15:00,71.14,261.77,204.11
9/30/2009 16:00,73.83,234.85,181.09
9/30/2009 17:00,73.33,198.26,129.27
9/30/2009 18:00,73.93,140.55,53.38
9/30/2009 19:00,73.75,124.13,20.04
9/30/2009 20:00,73.49,112.33,38.71
9/30/2009 21:00,72.44,113.27,47.55
9/30/2009 22:00,71.42,112.18,33.89
9/30/2009 23:00,71,111.39,70.27
10/1/2009 0:00,70.62,110.9,175.54
```

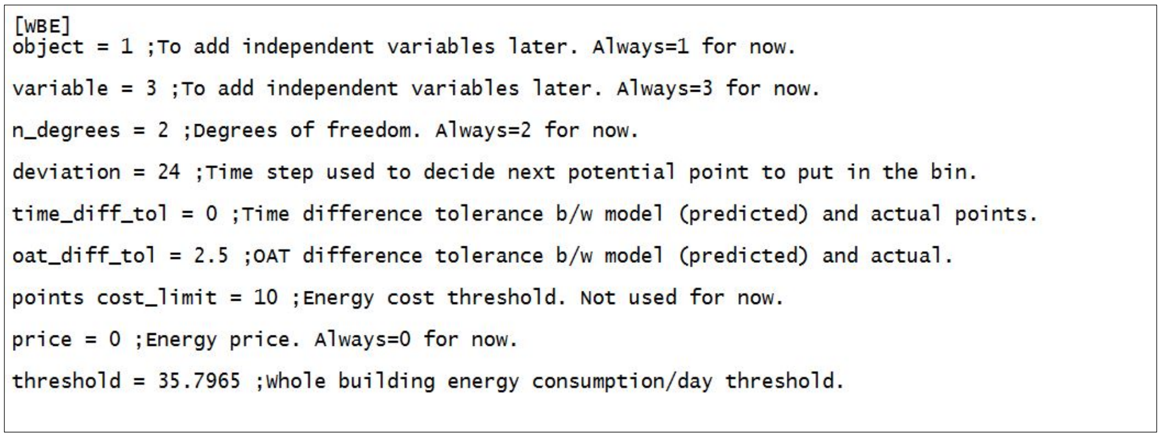
Figure 13. Example of the main data file



```
Timestamp,OutsideAirTemp,ReturnAirTemp,MixedAirTemp,CoolCall,CompressorStatus,HeatingStatus,FanStatus,Damper
5/19/2012 6:00,48.902,56.43727273,58.68472222,0,0,0,0,0
5/19/2012 7:00,51.12316667,59.47933333,59.58916667,0,0,0,0,0
5/19/2012 8:00,54.70866667,61.1625,64.34266667,0,0,0,0,0
5/19/2012 9:00,60.05372881,66.05525424,66.46474576,0,0,0,0,1.059322034
5/19/2012 10:00,65.98833333,70.66783333,69.93766667,0,0,0,1,12.29166667
5/19/2012 11:00,71.89383333,72.1775,72.66266667,0,0,0,1,12.5
5/19/2012 11:44,75.98,73.47733333,74.73844444,0,0,0,1,12.5
5/19/2012 12:00,77.59733333,73.79866667,75.04066667,1,1,0,1,6.98678
5/19/2012 13:00,79.63283333,73.99533333,75.607,1,1,0,1,6.648181667
5/19/2012 14:00,83.023,73.821,76.05816667,1,1,0,1,4.29151
5/19/2012 15:00,84.58783333,74.11366667,76.598,1,1,0,1,0.015793333
5/19/2012 15:26,85.98037037,74.51,77.08111111,1,1,0,1,1.78567037
5/19/2012 15:52,86.42,74.96,77.29,0,0,0,0,0
5/19/2012 16:00,86.34666667,74.61,77.315,1,1,0,1,5.410114286
5/19/2012 17:00,85.62283333,74.2685,77.05366667,1,1,0,1,5.5556
5/19/2012 18:00,84.836,74.12133333,76.87783333,1,1,0,1,5.5556
5/19/2012 19:00,83.14233333,74.27316667,76.41,1,1,0,1,5.5556
5/19/2012 20:00,79.44283333,74.06583333,75.33466667,1,1,0,1,0.750805
5/19/2012 21:00,74.28183333,73.77566667,73.71883333,1,1,0,1,0
5/19/2012 21:11,70.44166667,73.475,71.68666667,1,1,0,1,17.89164167
5/19/2012 21:20,69.34111111,74.08111111,70.02222222,1,0,0,1,82.00225556
5/19/2012 21:23,69.94333333,74.53,69.31333333,1,1,0,1,94.07406667
5/19/2012 21:29,69.98,74.16833333,69.37166667,1,0,0,1,91.1111
5/19/2012 21:58,70.4462069,74.03172414,69.96172414,1,1,0,1,100
5/19/2012 22:00,69.99,73.83,69.35,1,0,0,1,91.1111
5/19/2012 22:07,69.8775,73.7525,69.22875,1,0,0,1,91.1111
5/19/2012 22:11,70.14,73.8725,69.72,1,1,0,1,100
5/19/2012 22:13,70.02,73.71,69.44,1,0,0,1,95.55555
5/19/2012 22:17,69.9175,73.61,69.225,1,1,0,1,93.333325
5/19/2012 23:00,68.0597619,73.58142857,67.0002381,1,0,0,1,94.28570714
5/20/2012 0:00,66.36733333,73.31016667,65.55216667,1,0,0,1,100
5/20/2012 1:00,63.42783333,72.47583333,62.0895,1,0,0,1,100
5/20/2012 1:27,60.45642857,71.92357143,59.04035714,1,0,0,1,100
5/20/2012 2:00,57.990625,66.0428125,64.605625,0,0,0,0,1.909725
5/20/2012 3:00,54.2305,57.67016667,62.20583333,0,0,0,0,0
```

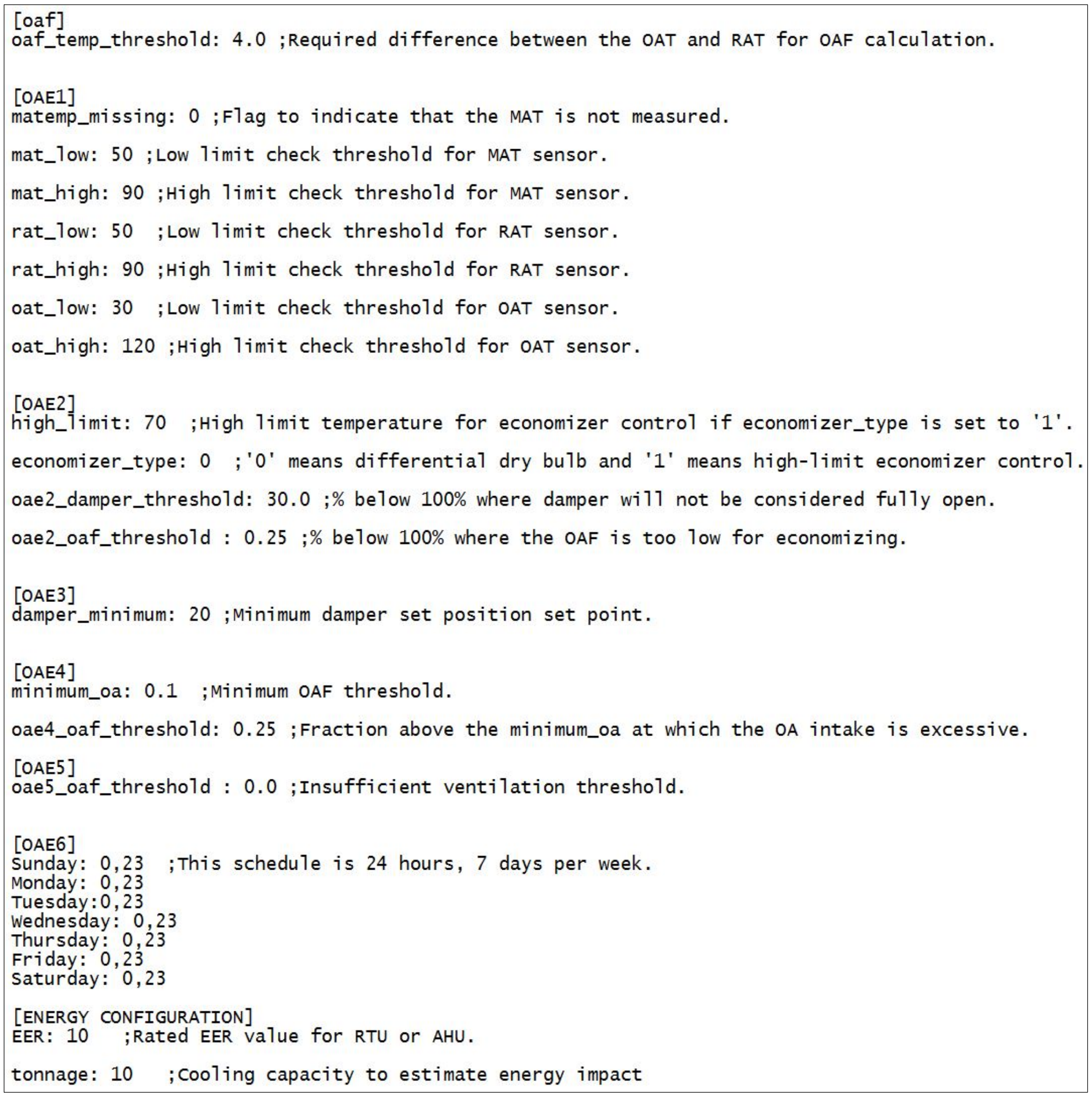
Figure 14. Example of the outside air economizer data file

Figures 15 and 16 show sample configuration files for the whole-building energy anomaly detection algorithm and the outside air economizer fault detection and diagnostic algorithm.



```
[WBE]
object = 1 ;To add independent variables later. Always=1 for now.

variable = 3 ;To add independent variables later. Always=3 for now.

n_degrees = 2 ;Degrees of freedom. Always=2 for now.

deviation = 24 ;Time step used to decide next potential point to put in the bin.

time_diff_tol = 0 ;Time difference tolerance b/w model (predicted) and actual points.

oat_diff_tol = 2.5 ;OAT difference tolerance b/w model (predicted) and actual.

points cost_limit = 10 ;Energy cost threshold. Not used for now.

price = 0 ;Energy price. Always=0 for now.

threshold = 35.7965 ;Whole building energy consumption/day threshold.
```

Figure 15. Sample configuration file for whole-building energy anomaly detection algorithm



```
[oaf]
oaf_temp_threshold: 4.0 ;Required difference between the OAT and RAT for OAF calculation.


[OAE1]
matemp_missing: 0 ;Flag to indicate that the MAT is not measured.

mat_low: 50 ;Low limit check threshold for MAT sensor.

mat_high: 90 ;High limit check threshold for MAT sensor.

rat_low: 50  ;Low limit check threshold for RAT sensor.

rat_high: 90 ;High limit check threshold for RAT sensor.

oat_low: 30  ;Low limit check threshold for OAT sensor.

oat_high: 120 ;High limit check threshold for OAT sensor.


[OAE2]
high_limit: 70  ;High limit temperature for economizer control if economizer_type is set to '1'.

economizer_type: 0  ;'0' means differential dry bulb and '1' means high-limit economizer control.

oae2_damper_threshold: 30.0 ;% below 100% where damper will not be considered fully open.

oae2_oaf_threshold : 0.25 ;% below 100% where the OAF is too low for economizing.


[OAE3]
damper_minimum: 20 ;Minimum damper set position set point.


[OAE4]
minimum_oa: 0.1  ;Minimum OAF threshold.

oae4_oaf_threshold: 0.25 ;Fraction above the minimum_oa at which the OA intake is excessive.

[OAE5]
oae5_oaf_threshold : 0.0 ;Insufficient ventilation threshold.


[OAE6]
Sunday: 0,23  ;This schedule is 24 hours, 7 days per week.
Monday: 0,23
Tuesday:0,23
Wednesday: 0,23
Thursday: 0,23
Friday: 0,23
Saturday: 0,23

[ENERGY CONFIGURATION]
EER: 10   ;Rated EER value for RTU or AHU.

tonnage: 10   ;Cooling capacity to estimate energy impact
```

Figure 16. Sample configuration file for outside-air economizer fault detection and diagnostic algorithm

Computer Hardware Requirements

The hardware and operating system must be able to run Python and the associated libraries, as shown below. That is, if a distribution of Python is available for a particular machine, then there are no other host system requirements.

Installing the Execution Environment

The "execution environment" refers to the system tools used to run the OpenEIS reference algorithms. The following tools must be installed:

- Python v2.7 (http://www.python.org/getit/).
- SciPy v0.12 (http://www.scipy.org/scipylib/download.html).
- Note that a "full stack" distribution may be available; this includes both Python and SciPy, already configured to work together (http://www.scipy.org/install.html).
- Numpy v1.7 (http://sourceforge.net/projects/numpy/files/NumPy/1.7.1/). Note that this should be installed along with SciPy.
- Matplotlib v1.2 (http://matplotlib.org/downloads.html). Note that this should be installed along with SciPy.
- SQLite (http://www.sqlite.org/download.html)

We recommend installing Python first, and allowing the installers to put the libraries in their default locations.

If your machine already has Python, plus either "pip" or "easy_install," these should provide an easy method for installing the additional libraries.

Installing the OpenEIS Reference Code

Next, install the OpenEIS reference code. The entire package can be downloaded from http://eis.lbl.gov/openeis.html. Expanding the downloaded zip file yields a directory containing the complete set of code and sample input files.

The directory containing the reference suite can be installed in any convenient location in your directory tree. We recommend placing it along with your regular documents, for example, under the "My Documents" directory on Windows, or under the "Documents" directory on Mac OSX.

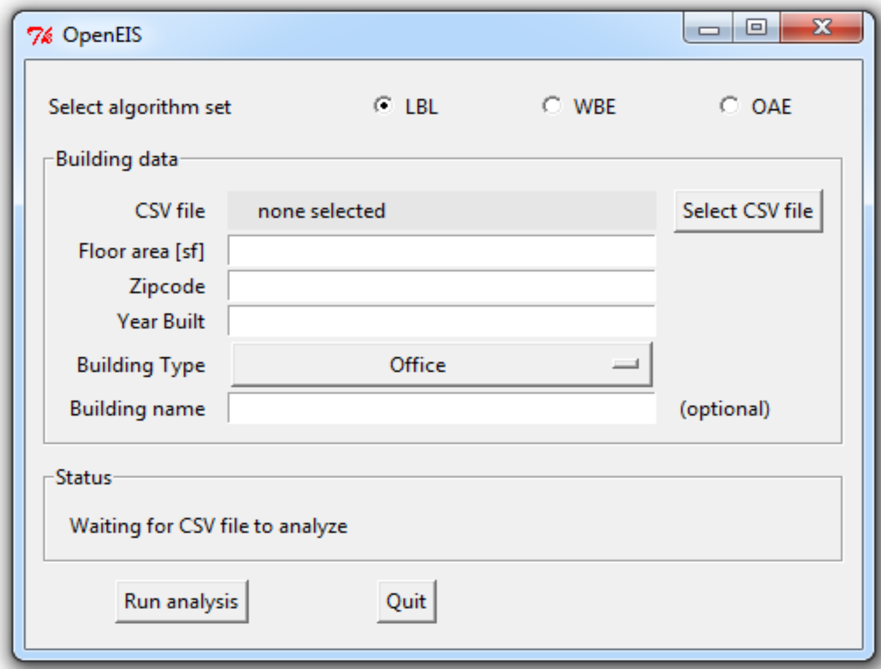
Similarly, the directory containing the reference suite can be given any name. The instructions that follow assume the folder is called "open_eis".

While the directory containing the reference suite can be relocated and renamed at will, this is not true of the "open_eis_lib" subdirectory. This subdirectory contains the scripts that implement the algorithms. Changing its name, the names of any of the files it contains, or the paths to any of the files it contains, will break the software distribution.

Code Execution

Next, run the algorithms on the sample data provided. To do so:

- Open a command-line window from which you can run Python. For example:
    - "Terminal" on Mac OS X.
    - "Command Prompt" on Windows (i.e., the DOS prompt).
    - "Terminal" on Ubuntu Linux.
    - Some Python distributions include a Python-specific command-line interpreter, distinct from shell utilities like "Terminal" and the DOS command prompt. For example, IDLE is the standard Python integrated development environment (http://docs.python.org/2/library/idle.html). However, the rest of these instructions assume you are using a conventional shell utility.
- Check that you can access Python. At the command-line prompt (>), enter:
  ```
  > python  -V
  ```
  The Python interpreter should respond with the version number. For example:
  ```
  > python  -V
  Python 2.7.5
  ```
  If not, consult your Python installation guide.
- Navigate to the OpenEIS installation directory. For example:
    - On Linux or Mac OS X:
      ```
      > cd  Documents/open_eis
      ```
    - On Windows:
      ```
      > cd  "My Documents\open_eis"
      ```
- Check for the Python script called "run_demo.py". For example, entering the following commands should show that the file exists:
    - On Linux or Mac OS X:
      ```
      > ls  run_demo.py
      ```
    - On Windows:
      ```
      > dir  run_demo.py
      ```
- Run the script:
  ```
  > python  run_demo.py
  ```
- This should bring up a small window, as shown below:



Select algorithm set [✓ LBL] WBE OAE

Building data

CSV file: none selected

Floor area [sf]:

Zipcode:

Year Built:

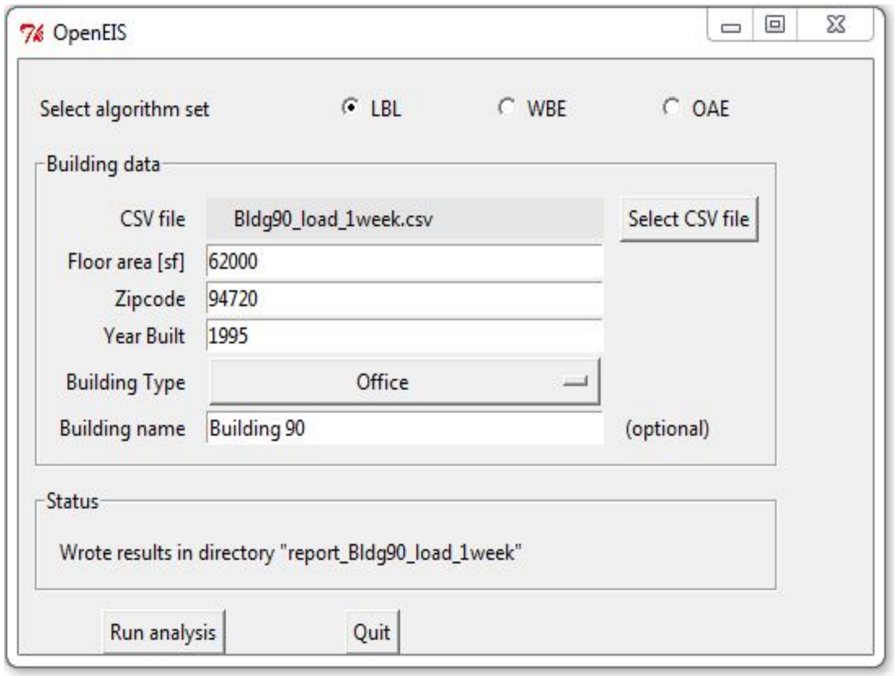
Building Type: Office

Building name: (optional)

Status

Waiting for CSV file to analyze

- To run all algorithms (with the exception of whole building energy anomaly detection or outside air economizer fault detection and diagnostics), select the "LBNL" radio button.
- Click the button to "Select CSV file". In the resulting dialog box, navigate to the subdirectory "sample_files/main_data_csv" in the "open_eis" directory. From that subdirectory, select one of the CSV files with "load" in its name.
- Fill in the building floor area (for the sample files with "Bldg90" in the name, 62000 square feet is appropriate, but the exact number is not necessary).
- Fill in the building zip code.
- Fill in the year built.
- If desired, fill in the building name.
- Click the "Run analysis" button.
- The algorithms should run against the data in the selected file. When done, the status text in the window should update to announce where to find the results. For example:


OpenEIS

Select algorithm set [✓ LBL] WBE OAE

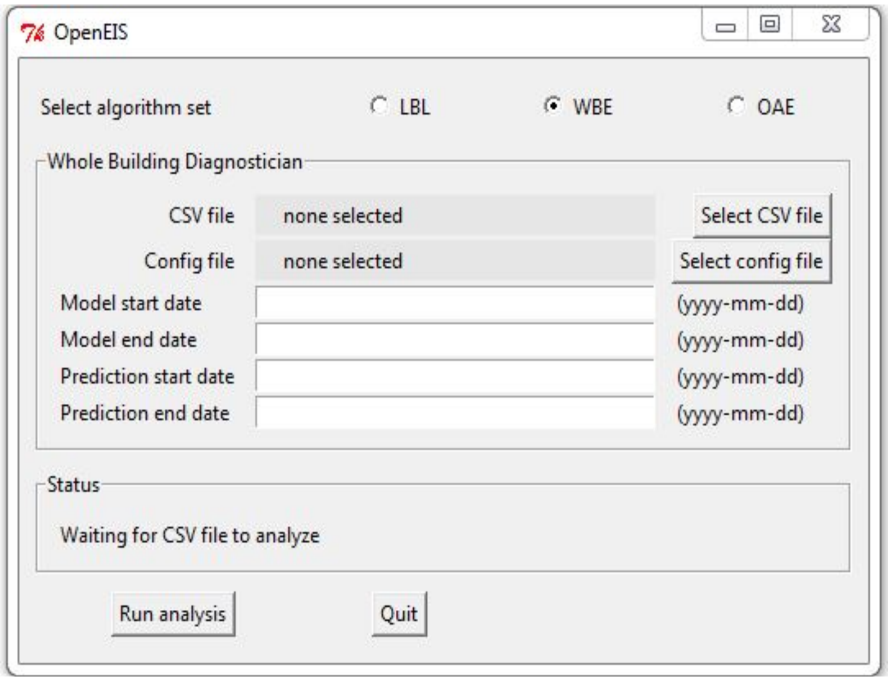
Building data

CSV file	Bldg90_load_1week.csv	Select CSV file
Floor area [sf]	62000	
Zipcode	94720	
Year Built	1995	
Building Type	Office	
Building name	Building 90	(optional)

Status

Wrote results in directory "report_Bldg90_load_1week"

- To run the whole building energy anomaly detection algorithm, select the "WBE" radio button. This should bring up a window, as shown below:


OpenEIS

Select algorithm set LBL [✓ WBE] OAE

Whole Building Diagnostician

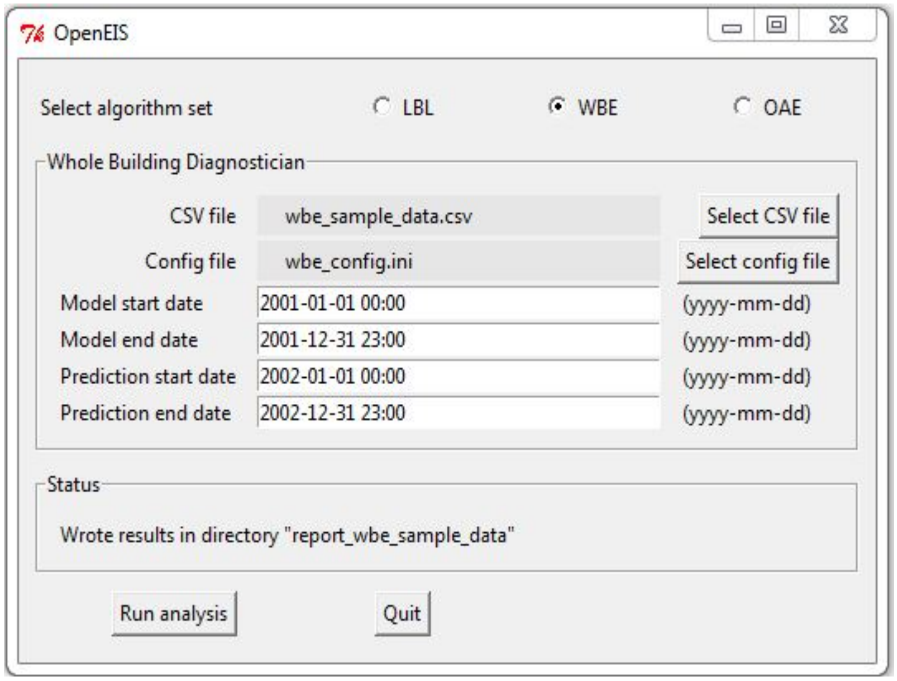
CSV file	none selected	Select CSV file
Config file	none selected	Select config file
Model start date		(yyyy-mm-dd)
Model end date		(yyyy-mm-dd)
Prediction start date		(yyyy-mm-dd)
Prediction end date		(yyyy-mm-dd)

Status

Waiting for CSV file to analyze

- Click the button to "Select CSV file." In the resulting dialog box, navigate to the subdirectory "sample_files/main_data_csv" in the "open_eis" directory. From that subdirectory choose "wbe_sample_data.csv."
- Click the button to "Select config file." In the resulting dialog box, navigate to the subdirectory "sample_files/config_file_ini" in the "open_eis" directory. From that subdirectory choose "wbe_config.ini."
- Fill in the Model start date: 2001-01-01 00:00

- Fill in the Model end date: 2001-12-31 23:00
- Fill in the Prediction start date: 2002-01-01 00:00
- Fill in the Prediction end date: 2002-12-31 23:00
- Click the “Run analysis” button.
- WBE should run against the data in the selected file. When done, the status text in the window should update to announce where to find the results. For example:

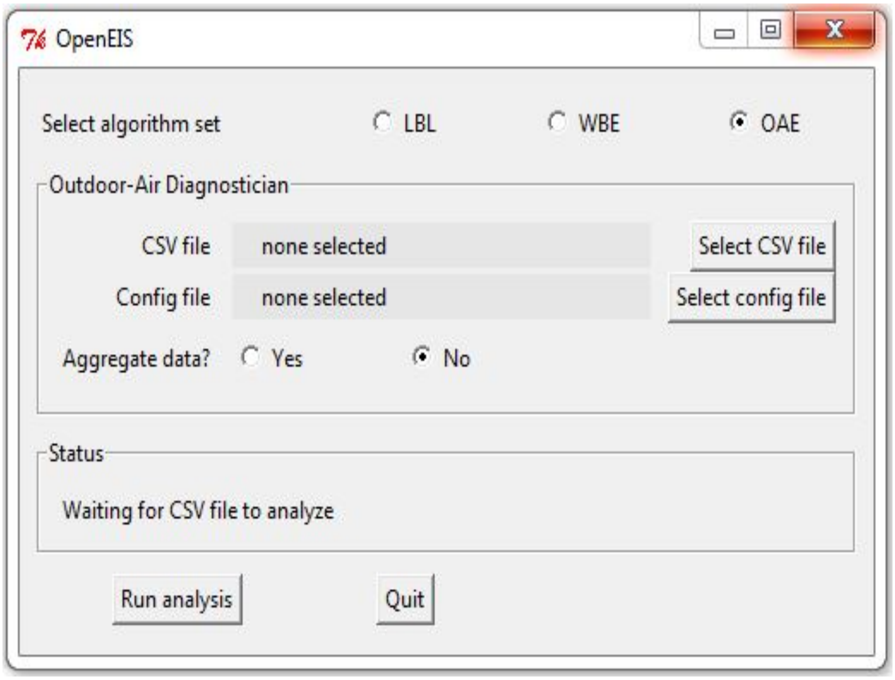


Select algorithm set: ○ LBL [✓ WBE] ○ OAE

CSV file: wbe_sample_data.csv
Config file: wbe_config.ini
Model start date: 2001-01-01 00:00 (yyyy-mm-dd)
Model end date: 2001-12-31 23:00 (yyyy-mm-dd)
Prediction start date: 2002-01-01 00:00 (yyyy-mm-dd)
Prediction end date: 2002-12-31 23:00 (yyyy-mm-dd)

Wrote results in directory "report_wbe_sample_data"

- To run the Outside air Economizer Diagnostician select the “OAE” radio button. This should bring up a window, as shown below:



Select algorithm set: LBL WBE [✓ OAE]

Outdoor-Air Diagnostician

CSV file: none selected

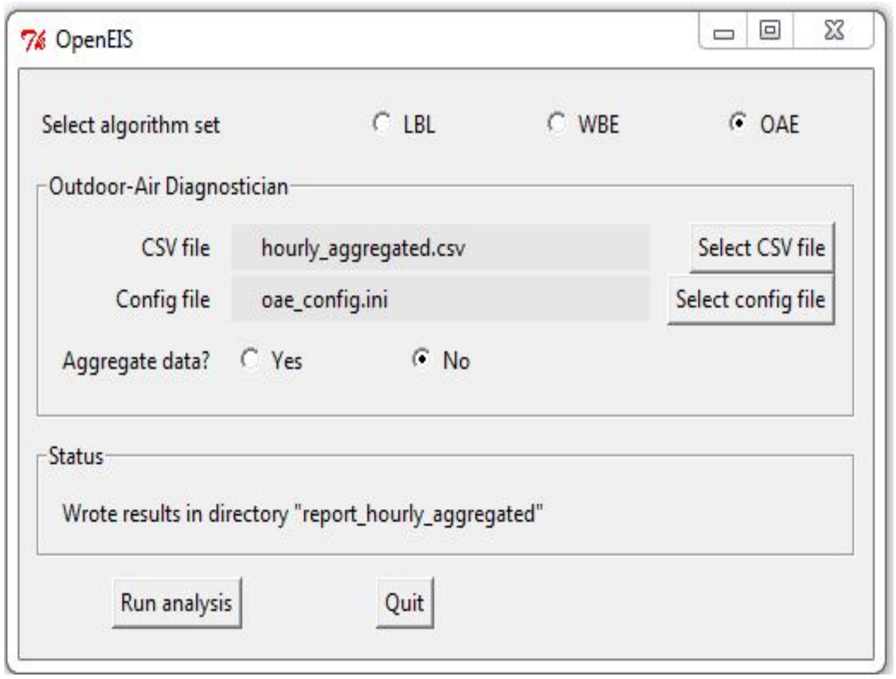
Config file: none selected

Aggregate data? Yes [✓ No]

Status

Waiting for CSV file to analyze

- Click the button to "Select CSV file." In the resulting dialog box, navigate to the subdirectory "sample_files/oae_data_csv" in the "open_eis" directory. From that subdirectory choose "hourly_aggregated.csv."
- Click the button to "Select config file." In the resulting dialog box, navigate to the subdirectory "sample_files/config_file_ini" in the "open_eis" directory. From that subdirectory choose "oae_config.ini."
- In the "Aggregate data?" field select "No." When inputting sub-hourly data select "Yes" in the "Aggregate data?" field. The OAE will internally aggregate the data to hourly values.
- Click the "Run analysis" button.
- OAE should run against the data in the selected file. When done, the status text in the window should update to announce where to find the results. For example:



CSV file: hourly_aggregated.csv
Config file: oae_config.ini
Aggregate data? Yes [✓ No]

Wrote results in directory "report_hourly_aggregated"

The output from running the algorithms is a set of html reports located in the code directory. Open these files to see the results for your building.

Terms of Use

This source code is available for free public use under a 3-clause BSD (Berkeley Software Distribution) license; registration and agreement to the terms and conditions are the only requirements for download.

Attribution and Reporting

The OpenEIS reference source code and pseudo-code were developed under funding from the U.S. Department of Energy, Building Technologies Office. They are available to the public at no charge.

Your feedback is critical to tracking the impact of this work. Please take a moment to send the OpenEIS project team a short note to let us know how you are using these algorithms.

Guidance for Product Developers and Programmers

This section reviews the terms of use and technical details for developers and programmers who wish to adapt the OpenEIS reference source code or pseudo-code for use in products or services.

Terms of Use

This source code is available for free public use under a 3-clause BSD (Berkeley Software Distribution) license; registration and agreement to the terms and conditions are the only requirements for download.

Attribution and Reporting

The OpenEIS reference source code and pseudo-code were developed under funding from the US Department of Energy, Building Technologies Office. They are available to the public, at no charge.

Your feedback is critical to tracking the impact of this work – please take a moment to send the OpenEIS project team a short note to let us know how you are using these algorithms.

Technical Details

Informal pseudo-code for the core OpenEIS algorithms appears below. The pseudo-code provides a high-level picture of the underlying logic for each algorithm, and how each metric or graphic is developed.

Developers who wish to implement a more robust, user-friendly tool are encouraged to refer to the source code developed as part of the OpenEIS algorithms reference implementation. That source code includes:

- Implementations of the core algorithms.
- Implementations of auxiliary methods needed to run the core algorithms (for example, reading comma-separated values (CSV) files, formatting output, and cleaning up missing data).
- Specifications, in the form of unit tests, for the exact results expected from the core algorithms. For example, the unit test for the weather sensitivity metric checks that the results from the reference code match independent calculations of the expected answer. These same tests can be adapted to check re-implementations of the core algorithms.

In addition, the distribution contains several types of documentation to help navigate the code. At a low level, each module, plus major functions, use Python "docstrings" to embed a variety of information directly into the code:

- The types and meaning of input data (e.g., function arguments).
- The types and meaning of outputs (e.g., function return results).
- Notes and caveats that will provide insight into how to use the function.
- Enhancements that would make the module more useful in a feature-complete implementation of the OpenEIS algorithms.

At a higher level, and maintained separately from the code, is documentation showing the broader organization of the code, along with overviews of the functions and how they interface. This includes versions of the pseudo-code shown in the next section. See the "developer_doc" subdirectory.

Pseudo-code

This section provides pseudo-code for the reference algorithms. The pseudo-code gives an overview of how to implement the algorithms, using high-level English language-like descriptions.

Time Series Load Profiling

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).

Identify the data of interest:
  - Take the last single month of data.  This ensures that the data are visually
    distinct.

Make the load profile plot:
  - Make an x-y line graph of y=last-month-loads as a function of
    x=last-month-times.

Done.
```

Heat Map

```
Get inputs:
  - times, vector of times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).

Assume:
  - Data are collected at the same time every day.

Identify the data of interest:
  - Take the last year of data.  This ensures that the data are visually distinct.

Break up the data into rows, each representing one day:
  - Reshape times into an array timesByDay, each of whose rows corresponds
    to one calendar day, with the date increasing in higher-numbered rows.  Use
    the local time zone to determine transitions between days.  Each column of
    timesByDay should correspond to a particular time of day.  Note this may
    require special padding for the first and last rows, if times does not
    start or end exactly at midnight.  If padding is needed, pad with a special
    NAN (not-a-number) indicator.
  - Reshape loads into an array loadsByDay, using the same row breaks and
    row padding as for timesByDay.

Make the heat map:
  - Define a color mapping from a power to a color.  Details include the spectrum
```

```
    of colors to be used, the min/max range of the color bar, and bin sizes for
    the color bar.
  - Make a density map, a binned x-y color map with x given by the time-of-day
    in timesByDay, and with y given by the dates in timesByDay.  The color
    of each cell is defined by applying the color mapping to the appropriate
    entry in loadsByDay.

Done.
```

Energy Signature

```
Get inputs:
  - oats, vector of outside air temperatures (float).
  - loads, vector of corresponding power data (float).

Identify the data of interest:
  - Take the most recent year of data.  This ensures that the data are visually
distinct.

Make the energy signature plot:
  - Make an x-y scatter plot, showing y = loads as a function of x = oats.

Done.
```

Load Duration Curve

```
Get inputs:
  - loads, vector of power data (float).
  - asPercent, flag indicating how to format the x-axis of the graph
    (boolean).  "True" means to express the duration as a percent of time.
    "False" means to express duration as the number of observations.

Identify the data of interest:
  - Take the most recent year of data. This limits the likelihood the data spans
different operational regimes (schedules, internal loads, etc…).


Assume:
  - The loads were recorded at uniform intervals.


Sort the loads:
  - Find sortedLoads by sorting loads in reverse order, i.e., from largest
    to smallest.  For example, if
    loads = (1, 3, 2, 4)
    then
    sortedLoads = (4, 3, 2, 1)


Find values for the x-axis:
  - Set LoadCt to the number of entries in loads.
  - if( asPercent is “True” ):
    * Set durs to a sequence of evenly-spaced percentages, from 0 to 100,
      with loadCt percentages in the sequence.
    * Note that this may require finding the step change from one number in
      the sequence to the next.  If so, then find
      step = 100 / (loadCt - 1)
      When doing this calculation, be sure to avoid integer division.
      For example, if
      loadCt = 4
      then
      step = 100/3 = 33.3333
      However, integer division may give 100 / 3 = 33.
  - else:
    * Set durs to the sequence of integers from 1 to loadCt.

Make the load duration curve:
  - Make an x-y line graph, showing y = sortedLoads as a function of
    x = durs.  As a practical matter, note that many plotting libraries do
    not require that durs be made explicit, when durs runs from 1 to
    loadCt (i.e., when asPercent is "False").
  - Typically the lower extent of the y-axis is fixed at zero power, in order
    to provide context for the range of power data.

Done.
```

Longitudinal Benchmarking

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).
  - areaFt2, floor area of corresponding space [ft^2] (float).

Assume:
  - Data include at least two years of observations.

Aggregate power data into annual energy intensity:
  - Separate loads into subsets of data that are 12 months long.  Mark years
    so that the last year ends on the last day in times.  For example, if the
    last observation is on 12-June, then every year should end on 12-June.
  - Set years to an empty list.
  - Set yearlyEnergyIntensities to an empty list.
  - For each year currYear, call the appropriate data currYearLoads:
    * Set currYearEnergy to the time integral of currYearLoads.
    * Set currYearIntensity to the energy intensity, i.e., to
      currYearEnergy / areaFt2.
    * Append currYear to years.
    * Append currYearIntensity to yearlyEnergyIntensities.

Make the longitudinal benchmarking plot:
  - Make a bar chart, showing y = yearlyEnergyIntensities as a function of
    category = years.

Done.
```

Weather Sensitivity

Weather sensitivity is calculated by finding the Spearman rank correlation coefficient, as follows:

```
Get inputs:
  - xValues and yValues, two vectors of values (float).  For weather
    sensitivity, xValues are the outside air temperatures, and yValues
    are the corresponding loads.  However, note that interchanging xValues
    and yValues will still give the same correlation coefficient.

Identify the data of interest:
  - Take the most recent year of data. This limits the likelihood the data spans
different operational regimes (schedules, internal loads, etc…).

Assume:
  - Both xValues and yValues have the same number of entries.
  - Any missing or corrupted entries in xValues and yValues have
    been marked as NAN (not-a-number).  These entries will be excluded from
    the analysis.

Mark pairs of entries for exclusion:
  - Set valCt to the number of entries in both xValues and yValues.
  - Set nanLocs to an array of valCt boolean values ("T" or "F").  The
    entry at each position in nanLocs indicates whether either xValues or
    yValues has a NAN in the corresponding location.  For example, if
    xValues = (1, 2, NAN, 4, NAN)
    yValues = (51, NAN,  53, 54,NAN)
    then
    nanLocs = (F, T, T, F, T)
  - Write a NAN to every position in xValues and yValues for which
    nanLocs is "T".  In the example above, this results in
    xValues = (1, NAN, NAN,  4, NAN)
    yValues = (51, NAN, NAN, 54, NAN)

Rank the remaining entries:
  - Set xRanks = rankForSpearman(xValues).
  - Set yRanks = rankForSpearman(yValues).
  - Note the pseudo-code for subprogram rankForSpearman() appears below.
  - Note both xRanks and yRanks are vectors containing valCt integers.
    For any valid index ii, the entry xRanks[ii] gives the ranking that
    xValues[ii] would have if sorted into ascending order.  Valid ranks run
    from 1 to valCt, with the following exceptions: (1) equal values receive
    the mean rank of those values; and (2) NAN entries receive a rank 0.

Subtract out the mean ranks:
  - Set xRanksZeroMean to xRanks minus the mean rank of xRanks.  Note
    that xRanks may contain zeros, due to NAN entries in xValues.  Exclude
    these zeros when finding the mean rank.  For example, if xRanks has
    valCt = 50 but two NAN entries, then the mean rank is the sum of the
    entries in xRanks, divided by 48.
  - Set yRanksZeroMean to yRanks minus the mean rank of yRanks.


Find the Spearman rank correlation coefficient:
  - Set cosineFactor to the inner (dot) product of xRanksZeroMean with
    yRanksZeroMean.
  - Set xMagSq to the inner (dot) product of xRanksZeroMean with itself.
  - Set yMagSq to the inner (dot) product of yRanksZeroMean with itself.
  - Find the Spearman coefficient using:
    spearmanCoeff = cosineFactor / sqrt(xMagSq * yMagSq)
  - Note that the Spearman coefficient is the Pearson coefficient of the
    rank vectors xRanks and yRanks.
  - Note that a correct calculation yields -1 <= spearmanCoeff <= 1.

Return spearmanCoeff.

Done.
```

The “rankForSpearman” Subprogram for the Spearman Algorithm

```
Get inputs:
  - values, vector containing valCt numbers (float).

Assume:
  - Any entries in values that should be excluded from the ranking
    have been marked as NAN (not-a-number).

Find the sorted order of entries in values:
  - Set srtdToActIdx to a vector of valCt indices that would sort values,
    from smallest to largest.  That is, srtdToActIdx[ii] should give the index
    of the iith entry in a sorted version of values.
  - Duplicate entries in values may be sorted in any order.  That is, it is
    not necessary to perform a "stable sort" that preserves the original order
    of duplicate entries in values.
  - NAN entries in values should be excluded from the main sequence of sorted
    values.  In practice, the sorting routine may treat NAN entries as larger
    (or smaller) than floating-point numbers, thus placing them at the end (or
    beginning) of srtdToActIdx.  The examples below assume that NAN entries
    sort to the highest position.
  - For example, if
    values = (6.6, 1.1, 3.3, NAN, 1.1)
    then
    srtdToActIdx = ( 1, 4, 2, 0, 3)
    is a possible ranking.  Consider ii = 3.  Since srtdToActIdx[3] = 0, it
    follows that values[0] = 6.6 would be at index 3 in a sorted version of
    values.  Note that there is one other acceptable ranking, due to the
    duplicated entry 1.1 in values.  Also note that, in programming languages
    that index arrays from 1 (such as R or Fortran), the entries in
    srtdToActIdx should be one greater than shown in the example.
  - Note that merely sorting values is not helpful, because the simple act of
    sorting does not retain information about which index of the original
    values vector provided each entry in the sorted vector.
  - In practice, some high-level programming environments provide sorting
    routines that return the indices needed to sort a vector; this is exactly
    the required srtdToActIdx.

Find the average rank order of each non-NAN entry in values:
  - Initialize ranks as a vector of valCt entries, all equal to 0.
  - Set lastVal to values[srtdToActIdx[0]].  Note this should be the
    smallest entry in values.
  - Set startRunIdx to 0.
  - for currIdx running from 1 to valCt-1 (i.e., for every entry after the first):
    * Set currVal to the corresponding entry in the sorted vector, i.e.,
      to values[srtdToActIdx[currIdx]].
    * if( currVal is NAN ):
      - Stop looping over currIdx.
    * if( currVal is different from lastVal, including if lastVal is NAN ):
      - Find the mean rank that should be assigned to indices startRunIdx
        through currIdx-1, inclusive.  The sorted rank at currIdx-1 is
        currIdx, so find the mean rank as
        meanRank = 0.5 * (startRunIdx + currIdx + 1)
        Note that in a language that indexes arrays from 1, the sorted rank
        at currIdx-1 is currIdx-1, so the mean rank is
        meanRank = 0.5 * (startRunIdx + currIdx - 1)
      - while( startRunIdx < currIdx ):
        * Set ranks[srtdToActIdx[startRunIdx]] to meanRank.
        * Increment startRunIdx by 1.
      - Set lastVal to currVal, in order to mark the start of a new run
        of equal values.  Note that startRunIdx should already be equal to
        currIdx, due to the while-loop above.
  - Assign ranks to the last entries inspected in the loop above:
    * Set meanRank to the mean rank, using the same formula shown above.
    * while( startRunIdx < currIdx ):
      - Set ranks[srtdToActIdx[startRunIdx]] to meanRank.
      - Increment startRunIdx by 1.
  - Here, every entry of ranks should have the rank of the corresponding
    entry in values, with equal values assigned their mean rank, and with
    NAN values assigned a rank of 0.  Continuing the example above, if
    values = (6.6, 1.1, 3.3, NAN, 1.1)
    then
    ranks = (4, 1.5, 3, 0, 1.5)
    Note that, unlike the entries in srtdToActIdx, ranks does not depend on
    whether the programming language indexes arrays from 0 or from 1.  Ranks
    always range from 1 (or larger, for the smallest non-NAN entry in values)
    to valCt (or less, for the largest non-NAN entry in values).

Return ranks.

Done.
```

Base-to-Peak Load Ratio

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).

Compute the base-to-peak load metric:
  - For each day in times, find the base-to-peak ratio of the loads:
    * Set dayBase to the 5th percentile of loads for the day.
    * Set dayPeak to the 95th percentile of loads for the day.
    * Set dayBPRatio to dayBase / dayPeak.
  - Average across days:
    * Set aveDayBPRatio to the average of the dayBPRatio values.

Return aveDayBPRatio.

Done.
```

Peak Load Benchmarking

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).
  - areaFt2, floor area of corresponding space [ft^2] (float).

Calculate statistics:
  - Set peakLoad to the maximum value in loads.
  - Set peakLoadIntensity to peakLoad / areaFt2.

Return peakLoadIntensity.
  - TODO: Code currently reports peakLoad, not peakLoadIntensity.

Done.
```

Load Variability

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).

Assume:
  - times are hourly observations.  That is, each day has 24 observations.
    If the original data were recorded at finer granularity, then the loads
    represent the average power for the hour in question.

Find the load variability for each unique timeOfDay in times:
  - Set todLoads to those entries from loads that were recorded at one
    unique timeOfDay from times.
  - Set todCt to the number of observations in todLoads.
  - Set todAve to the average of todLoads.
  - Set todSumSqDev to the sum of the squares of the differences between
    todLoads and todAve.
  - Set todStdDev to the corrected sample standard deviation of the todLoads,
    that is, to the square root of (todSumSqDev / (todCt - 1)).
  - Set todVar to the variability of todLoads, that is, to todStdDev / todAve.

Find the average of the daily load variabilities:
  - Set aveTodVar to the average of the todVar values.

Return aveTodVar.
```

```
Done.
```

Cross-Sectional Benchmarking

```
Get inputs:
  - times, vector of date-times (typically a time-specific format).
  - loads, vector of power data recorded at times (float).
  - areaFt2, floor area of corresponding space [ft^2] (float).
  - zipcode, 5-digit postal code of corresponding space (integer).
  - bldgType, type of building from enumerated list (string).
  - bldgYear, year when the building is constructed (integer).

Assume:
  - Data include at least two years of observations.
  - Webservice default values, if applicable, are used, in order
    to limit the number of inputs required from direct users.
  - Default value for bldgType is ‘Office’.

Aggregate power data into annual energy intensity:
  - Separate loads into subsets of data that are 12 months long.
    Mark years so that the last year ends on the last day in times.
    For example, if the last observation is on 12-June, then every year
    should end on 12-June.
  - Set years to an empty list.
  - Set yearlyEnergyIntensities to an empty list.
  - For each year currYear, call the appropriate data currYearLoads:
    * Set currYearEnergy to the time integral of currYearLoads.
    * Append currYear to years.

Generate XML files:
  - Replace keywords in the targetFinderdata XML template with the
    corresponding data inputs.
    * Set property-zipcode to zipcode.
    * Set property-floor-area to areaFt2.
    * Set property-year-built to bldgYear.
    * Set property-type to bldgType.
    * Set meter-usage to yearlyEnergy.
    * Set meter-type to ‘Electric’ or ‘Natural Gas’.
    * Set meter-energy-units to ‘kWh (thousand Watt-hours)’ or ‘kBtu (thousand Btu)’.
 - Return targetFinderdata XML-formatted strings.

Retrieve Energy Star Score from web services:
 - Pass propertyUse string to generate propertyUse in Portfolio Manager.
 - Query the energy star ‘score’ and append to PMMetrics.

Make the cross-sectional benchmarking plot:
  - Make a single-bar chart, showing y = PMMetrics[‘designScore’].

Done.
```

Other Summary Electric Load Statistics, Displayed in the Report Table

```
Get inputs:
  - *times*, vector of date-times (typically a time-specific format).
  - *loads*, vector of power data recorded at *times* (float).

Calculate statistics:
  - For each day in *times*, find the metrics of interest:
    * Set *dayBase* to the 5th percentile of *loads* for the day.
    * Set *dayPeak* to the 95th percentile of *loads* for the day.
    * Set *dayRange* to *dayPeak* - *dayBase*.
  - Average across days:
    * Set *aveDayBase* to the average of the *dayBase* values.
    * Set *aveDayPeak* to the average of the *dayPeak* values.
    * Set *aveDayRange* to the average of the *dayRange* values.

Return *aveDayBase*, *aveDayPeak*, and *aveDayRange*.

Done.
```

Whole-Building Energy Anomaly Detection

Algorithm inputs:

$T_{out,\ base}$ – Baseline outdoor-air temperature (hourly average)
$T_{out,\ post}$ – Post-baseline outdoor-air temperature (hourly average)
Timestamp – date and time (hourly)
WBE – Whole building energy (hourly energy consumption in kWh)
$t_{start,baseline}$ – Indicates start time to be used to develop baseline model (specified in configuration file)
$t_{end,baseline}$ – Indicates end time to be used to develop baseline model (specified in configuration file)
$t_{start,post}$ – Indicates start time for post-baseline (specified in configuration)
$t_{end,post}$ – Indicates start time for post-baseline (specified in configuration)
$T_{Threhsold}$ – ± value used when mapping post-baseline T_{out} to a baseline bin to calculate expected energy consumption

Algorithm steps:

1. Read the baseline data from the csv file:
   * The Whole Building Energy Diagnostician uses $T_{out,base}$ as an independent variable (the hour of week (HOW) is implicitly assumed to be a second independent variable when creating the model) to develop a model for predicting a building's energy consumption (dependent variable). In the future, the WBE Diagnostician will have the capability to use multiple independent parameters (e.g., outdoor-air relative humidity).

2. Obtain user inputs:
   * The user inputs the start and end date for the baseline period and post-baseline period via the OpenEIS UI.

3. Read the post-baseline data to predict the expected hourly energy consumption:
   * The post baseline will have an hour of the week (HOW) and a $T_{out,post}$ associated with each data entry.

$$\mathrm{WBE_{expected}} = f(\mathrm{T_{out,post}} \pm \mathrm{T_{threshold}}, \mathrm{HOW})$$

A bin method is used to predict the expected consumption based on the baseline data. For each hourly prediction, the data bin is created based on the outdoor temperature and the hour of the week.

The predicted energy consumption is the median energy consumption from the bin associated with the corresponding post-baseline HOW for which the energy prediction is being made and the outdoor temperature.

4. Calculate the median, root mean square error (RMSE), and mean bias error (MBE) for each bin:
   * RMSE, MBE along with the median are used to estimate the uncertainty associated with the prediction of the expected energy consumption. The median is the estimate or prediction of the consumption for that hour.

5. Calculate the daily predicted energy consumption and the daily actual energy consumption:
   * Using the hourly predicted energy consumption for the building to calculate the daily energy consumption. The daily energy consumption is the sum the of the hourly energy consumption for each hour within that day.

6. Compare the predicted energy consumption and the actual energy consumption and compute the Energy Consumption Index (ECI) as a ratio of actual and predicted energy consumption:
   * ECI value along with the uncertainty can be used to detect deviations of actual consumption from its expected normal.

7. Create output graphs and tables.

Outside-Air Economizer Fault Detection and Diagnostics

Nomenclature (Order of Appearance)

N_{data} – Number rows of data in the CSV file containing the input data.
$FanStat_i$ – Fan status at time step i.
$OAE1_i$ – Result for OAE1 at time step i.
MAT_i – Mixed-air temperature sensor reading at time step i.
OAT_i – Outdoor-air temperature sensor reading at time step i.
RAT_i – Mixed-air temperature sensor reading at time step i.
F_{MAT} – Flag indicating that the AHU or RTU does not measure mixed-air temperature.
$Compressor_i$ – Compressor status at time step i. A True value indicates the unit is mechanically cooling.
$Heating_i$ – Heating status at time step i. A True value indicates the unit is heating.
MATLOW – Low-end threshold for MAT sensor.
MATHIGH – High-end threshold for MAT sensor.
OATLOW – Low-end threshold for OAT sensor.
OATHIGH – High-end threshold for OAT sensor.
RATLOW – Low-end threshold for RAT sensor.
RATHIGH – High-end threshold for RAT sensor.
$OAE2_i$ - Result for OAE2 at time step i.
$CoolCall_i$ – Cooling call status at time step i. A True value indicates there is call for cooling.
$Damper_i$ – Damper signal at time step i.
EconomizerType – 0 indicates differential dry bulb and 1 indicates high limit economizer.
HighLimit – The high limit temperature for economizing if EconomizerType = 1.
OAE2 Damper threshold – Damper open to 100% threshold (default value: 5).
OAE2 Temperature threshold – Threshold to determine if conditions are OK for OAF calculation.
OAE2 OAF threshold – Value used to determine if nearly 100% OA is used (default: 0.25).
OAF_i – The calculated OAF at times step i.
abs() – Function that returns the absolute value of a numeric input.
$OAE3_i$ – Result for OAE3 at time step i.
MinimumDamper – Minimum outdoor-air damper position set point for unit.
$OAE4_i$ – Result for OAE4 at time step i.
OAE4 Temperature threshold – Threshold to determine if conditions are OK for OAF calculation.
MinimumOA – Minimum required OA percent (default: 0.20)
OAE4 OAF threshold – Value used to determine if nearly 100% OA is used (default: 0.25).
$OAE5_i$ – Result for OAE5 at time step i.
$OAE6_i$ – Result for OAE6 at time step i.
$SchedHour_i$ – Numeric value of hour parsed from Timestamp input
$TimeStamp_i$ – Timestamp for data at time step i.
weekday() – function takes in timestamp and return numeric value for day-of-week. Sunday – Saturday is 0-6.
day – numeric value returned by weekday() within OAE6 diagnostic.
weekSchedule – list of lists that contain the start and end schedule for each day-of week
daySchedule - list with start and end time for specific day of the week.
StartHour – Start-up hour for unit (configurable).
EndHour – End hour for unit (configurable).
hour() – Function that returns the numeric hour value of a time object.

OAE1: Air-side Temperature Sensor (outdoor-, return-, or mixed-air) Diagnostic

```
FOR i = 1 to $N_{data}$:  (loop over each data entry)
    IF $FanStat_i$ = -99:  (-99 is missing data indicator)
        THEN $OAE1_i$ = 27 (FanStat for ith row is missing)
        NEXT i
    ENDIF
    IF FanStat_i Is True:
        IF (MAT_i Or OAT_i Or RAT_i) = -99:
            THEN OAE1_i = 27 (Data for i^th row is missing)
            NEXT i
        ENDIF
        IF F_MAT is True And (Compressor_i Or Heating_i) Are True:
            THEN OAE1_i = 22 (Conditions not favorable for diagnostic)
            NEXT i
        ENDIF
        IF ((MAT_i < MATLOW Or MAT_i > MATHIGH:
            THEN OAE1_i = 23 (Mixed-air temperature sensor outside expected range)
            NEXT i
        ENDIF
        IF ((RAT_i < RATLOW Or RAT_i > RATHIGH:
            THEN OAE1_i = 24 (Return-air temperature sensor outside expected range)
            NEXT i
        ENDIF
        IF ((OAT_i < OATLOW Or OAT_i > OATHIGH:
            THEN OAE1_i = 25 (Outside-air temperature sensor outside expected range)
            NEXT i
        ENDIF
       IF ((MAT_i < OAT_i And MAT_i < RAT_i) Or (MAT_i > OAT_i And MAT_i  > RAT_i):
            THEN OAE1_i = 21 (temperature sensor fault)
            NEXT i
        ELSE:
          THEN OAE1_i = 20 (No fault detected)
          NEXT i
       ENDIF
   ELSE:
        THEN OAE1_i = 29 (RTU is OFF)
        NEXT i
    ENDIF
ENDFOR
GOTO OAE2
```

OAE2: Not Economizing when the RTU Should

```
FOR i = 1 to N_data:  (loop over each data entry)
    IF FanStat_i = -99:  (-99 is empty data indicator)
        THEN OAE2_i = 37 (FanStat for i^th row is missing)
        NEXT i
    ENDIF
    IF FanStat_i Is True:
        IF (CoolCall_i Or Damper_i Or OAT_i Or RAT_i) = -99:
            THEN OAE2_i = 37 (Data for i^th row is missing)
            NEXT i
        ENDIF
        IF (CoolCall_i Is True And OAT_i < RAT_i And EconomizerType = 0) Or
             (CoolCall_i Is True And OAT_i < HighLimit And EconomizerType = 1):
            IF (100 – Damper_i < OAE2 Damper threshold):
                IF ((abs(OAT_i  - RAT_i > OAE2 Temperature threshold) and F_MAT Is False):
                    IF (1.0 – OAF_i < OAE2 OAF threshold And OAF_i < 1.25 And OAF_i > 0):
                        THEN OAE2_i = 30 (No Fault)
                        NEXT i
                    ElSEIF (1.0 – OAF_i > OAE2 OAF threshold And OAF_i < 1.25 And OAF_i >
0):
                        THEN OAE2_i = 32 (OAF is too low when economizing)
                        NEXT i
                    ELSE:
                        THEN OAE2_i = 38 (Damper is open but OAF was unexpected value)
                        NEXT i
                    ENDIF
                ELSEIF ((abs($OAT_i$  - $RAT_i$ > OAE2 Temperature threshold) And
($Compressor_i$ And                       $Heating_i$) Are False And $F_{MAT}$ Is TRUE ):
                    IF (1.0 – $OAF_i$ < OAE2 OAF threshold And $OAF_i$ < 1.25 And $OAF_i$ > 0):
                        THEN $OAE2_i$ = 30 (No Fault)
                        NEXT i
                    ElSEIF (1.0 – $OAF_i$ > OAE2 OAF threshold And $OAF_i$ < 1.25 And $OAF_i$ >
0):
                        THEN $OAE2_i$ = 32 (OAF is too low when economizing)
                        NEXT i
                    ELSE:
                        THEN $OAE2_i$ = 38 (Damper is open but OAF was unexpected value)
                        NEXT i
                    ENDIF
                ELSE:
                    THEN $OAE2_i$ = 36 (Damper is OK, conditions not favorable for OAF
calc.)
                    NEXT i
                ENDIF
            ELSE:
                THEN $OAE2_i$ = 33 (Damper should be open to economize)
                NEXT i
            ENDIF
        ELSE:
            THEN $OAE2_i$ = 31 (No Call for cooling the unit should not economize)
            NEXT i
        ENDIF
    ELSE:
        THEN $OAE2_i$ = 39 (RTU is OFF)
        NEXT i
    ENDIF
ENDFOR
GOTO OAE3
```

<u>OAE3: Economizing when the RTU should not</u>

```
FOR i = 1 to $N_{data}$:  (loop over each data entry)
    IF $FanStat_i$ = -99:  (-99 is missing data indicator)
        THEN $OAE3_i$ = 47 (FanStat for i$^{th}$ row is missing)
        NEXT i
    ENDIF
    IF $FanStat_i$ Is True:
        IF ($CoolCall_i$ Or $Damper_i$ Or $OAT_i$ Or $RAT_i$) = -99:
            THEN $OAE3_i$ = 47 (Data for i$^{th}$ row is missing)
            NEXT i
        ENDIF
        IF $CoolCall_i$ Is True:
            IF ($OAT_i$ > $RAT_i$ And EconomizerType = 0) Or
                 ($OAT_i$ > HighLimit And EconomizerType = 1):
                IF $Damper_i$ <= MinimumDamper:
                    THEN $OAE3_i$ = 40 (Damper is correctly at minimum)
                    NEXT i
                ELSE:
                    THEN $OAE3_i$ = 41 (Damper should be at minimum)
                    NEXT i
                ENDIF
            ELSE:
                THEN $OAE3_i$ = 43 (Conditions favorable for economizing)
                NEXT i
            ENDIF
        ELSE:
            IF $Damper_i$ <= MinimumDamper:
                THEN $OAE3_i$ = 40 (Damper is correctly at minimum)
                NEXT i
            ELSE:
                THEN OAE3$_i$ = 41 (Damper should be at minimum)
                NEXT i
            ENDIF
        ENDIF
    ELSE:
        THEN OAE3$_i$ = 49 (RTU is OFF)
        NEXT i
    ENDIF
ENDFOR
GOTO OAE4
```

OAE4: Excess Outdoor-air Ventilation

```
FOR i = 1 to N$_{data}$:  (loop over each data entry)
    IF FanStat$_i$ = -99:  (-99 is empty data indicator)
        THEN OAE4$_i$ = 57 (FanStat for i$^{th}$ row is missing)
        NEXT i
    ENDIF
    IF FanStat$_i$ Is True:
        IF (CoolCall$_i$ Or Damper$_i$ Or OAT$_i$ Or RAT$_i$ Or MAT$_i$) = -99:
            THEN OAE4$_i$ = 57 (Data for i$^{th}$ row is missing)
            NEXT i
        ENDIF
        IF (CoolCall$_i$ Is True And OAT$_i$ > RAT$_i$ And EconomizerType = 0) Or
             (CoolCall$_i$ Is True And OAT$_i$ > HighLimit And EconomizerType = 1):
            IF (Damper$_i$ > DamperMinimum):
                THEN OAE4$_i$ = 53 (Damper should be at minimum)
                NEXT i
            ENDIF
            IF ((abs(OAT$_i$  - RAT$_i$ > OAE4 Temperature threshold) and F$_{MAT}$ Is False):
                IF (OAF$_i$ – MinimumOA < OAE4 OAF threshold And OAF$_i$ < 1.25
                     And OAF$_i$ > 0):
                    THEN OAE4$_i$ = 50 (No Fault)
                    NEXT i
                ElSEIF (OAF$_i$ – MinimumOA > OAE4 OAF threshold And OAF$_i$ < 1.25
                    And OAF$_i$ > 0):
                    THEN OAE4$_i$ = 51 (Excess OA being brought in by RTU)
                    NEXT i
                ELSE:
                    THEN OAE4$_i$ = 58 (Damper is open but OAF was unexpected value)
                    NEXT i
                ENDIF
            ELSEIF ((abs(OAT$_i$  - RAT$_i$ > OAE4 Temperature threshold) And
                    (Compressor$_i$ And  Heating$_i$ ) Are False And F$_{MAT}$ Is TRUE):
                IF (OAF$_i$ – MinimumOA < OAE4 OAF threshold And
                     (OAF$_i$ < 1.25 And OAF$_i$ > 0)):
                    THEN OAE4$_i$ = 50 (No Fault)
                    NEXT i
                ElSEIF (OAF$_i$ – MinimumOA > OAE4 OAF threshold And
                     (OAF$_i$ < 1.25 And OAF$_i$ > 0)):
                    THEN OAE4$_i$ = 51 (Excess OA being brought in by RTU)
                    NEXT i
                ELSE:
                    THEN OAE4$_i$ = 58 (Damper is open but OAF was unexpected value)
                    NEXT i
                ENDIF
            ELSE:
                THEN OAE4$_i$ = 52 (Damper is OK, conditions not favorable for OAF calc.)
                NEXT i
            ENDIF
        ELSE
            THEN OAE4$_i$ = 56 (Unit should be economizing, No Fault)
            NEXT i
        ENDIF
    ELSE:
        THEN OAE4_i = 59 (RTU is OFF)
        NEXT i
    ENDIF
ENDFOR
GOTO OAE5
```

OAE5: Insufficient Outdoor-air Ventilation

```
FOR i = 1 to N_data:  (loop over each data entry)
    IF FanStat_i = -99:  (-99 is empty data indicator)
        THEN OAE5_i = 67 (FanStat for ith row is missing)
        NEXT i
    ENDIF
    IF FanStat_i Is True:
        IF (CoolCall_i OR Damper_i OR OAT_i OR RAT_i OR MAT_i) = -99:
            THEN OAE5_i = 67 (Data for ith row is missing)
            NEXT i
        ENDIF
        IF (CoolCall_i Is True And OAT_i > RAT_i And EconomizerType = 0) Or
             (CoolCall_i Is True And OAT_i > HighLimit And EconomizerType = 1):
            IF (Damper_i > DamperMinimum):
                THEN OAE4_i = 63 (Damper should be at minimum)
                NEXT i
            ENDIF
            IF ((abs(OAT_i  - RAT_i > OAE5 Temperature threshold) and F_MAT Is False):
                IF (MinimumOA - OAF_i > OAE5 OAF threshold And
                     ( OAF_i < 1.25 And OAF_i > 0)):
                    THEN OAE5_i = 61 (Insufficient OA ventilation)
                    NEXT i
                ElSEIF (MinimumOA - OAF_i < OAE5 OAF threshold And
                     (OAF_i < 1.25 And OAF_i > 0)):
                    THEN OAE5_i = 60 (No fault)
                    NEXT i
                ELSE:
                    THEN OAE5_i = 68 (Damper is open but OAF was unexpected value)
                    NEXT i
                ENDIF
            ELSEIF ((abs(OAT_i  - RAT_i > OAE4 Temperature threshold) And
                 (Compressor_i And Heating_i) Are False And F_MAT Is TRUE ):
                 IF (MinimumOA - OAF_i > OAE5 OAF threshold And
                     (OAF_i < 1.25 And OAF_i > 0)):
                    THEN OAE5_i = 61 (Insufficient OA ventilation)
                    NEXT i
                ElSEIF (MinimumOA - OAF_i < OAE5 OAF threshold And
                     (OAF_i < 1.25 And OAF_i > 0)):
                    THEN OAE5_i = 60 (No fault)
                    NEXT i
                ELSE:
                    THEN OAE5_i = 68 (Damper is open but OAF was unexpected value)
                    NEXT i
                ENDIF
            ELSE:
                THEN OAE5_i = 62 (Damper is OK, conditions not favorable for OAF calc.)
                NEXT i
            ENDIF
        ELSE
            THEN OAE5_i = 66 (Unit should be economizing (No Fault))
            NEXT i
        ENDIF
    ELSE:
        THEN OAE5_i = 69 (RTU is OFF)
        NEXT i
    ENDIF
ENDFOR
GOTO OAE6
```

OAE6: RTU Operating Outside of Schedule

```
FOR i = 1 to N_data:  (loop over each data entry)
    IF FanStat_i = -99 Or Compressor_i = -99:  (-99 is empty data indicator)
        THEN OAE6_i = 77 (FanStat or Compr for ith row is missing)
        NEXT i
    ENDIF
    day = weekday(TimeStamp_i)
    daySchedule = weekSchedule[day]
    StartHour = daySchedule [0]
    EndHour = daySchedule [1]
    SchedHour_i = hour(TimeStamp_i)
    IF (FanStat_i Or Compressor_i) Are True And
         (SchedHour_i < StartHour Or SchedHour_i > EndHour):
        THEN OAE6_i = 71 (RTU is operating outside of scheduled time).
        NEXT i
    ELSE:
        THEN OAE6_i = 70 (No Fault).
        NEXT i
    ENDIF
ENDFOR
END OAE
```