

User Guide: OpenEIS Reference Code

**Jessica Granderson
David Lorenzetti
Claudine Custodio**

Building Technology and Urban Systems Department
Lawrence Berkeley National Laboratory



Prepared for:
George Hernandez
DOE Building Technologies Office

September 30, 2013

Table of Contents

Purpose.....	1
Algorithms Overview.....	2
Time Series Load Profiling.....	2
Heat Maps.....	3
Energy Signature.....	4
Weather Sensitivity.....	5
Longitudinal Benchmarking.....	6
Peak Load Benchmarking.....	7
Base-to-Peak Load Ratios.....	8
Load Duration Curve.....	9
Load Variability.....	10
Data Requirements.....	11
Guidance for Direct Users.....	12
Data File Formatting Requirements.....	12
Computer Hardware Requirements.....	13
Installing the Execution Environment.....	13
Installing the OpenEIS Reference Code.....	14
Code Execution.....	14
Terms of Use.....	16
Attribution and Reporting.....	16
Guidance for Product Developers and Programmers.....	17
Terms of Use.....	17
Attribution and Reporting.....	17
Technical Details.....	17
Pseudo-code.....	19
Time Series Load Profiling.....	19
Heat Map.....	20
Energy Signature.....	20
Load Duration Curve.....	21
Longitudinal Benchmarking.....	22
Weather Sensitivity.....	23

The “rankForSpearman” Subprogram for the Spearman Algorithm	24
Base-to-Peak Load Ratio	26
Peak Load Benchmarking	26
Load Variability.....	27
Other Summary Electric Load Statistics, Displayed in the Report Table.....	27

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.

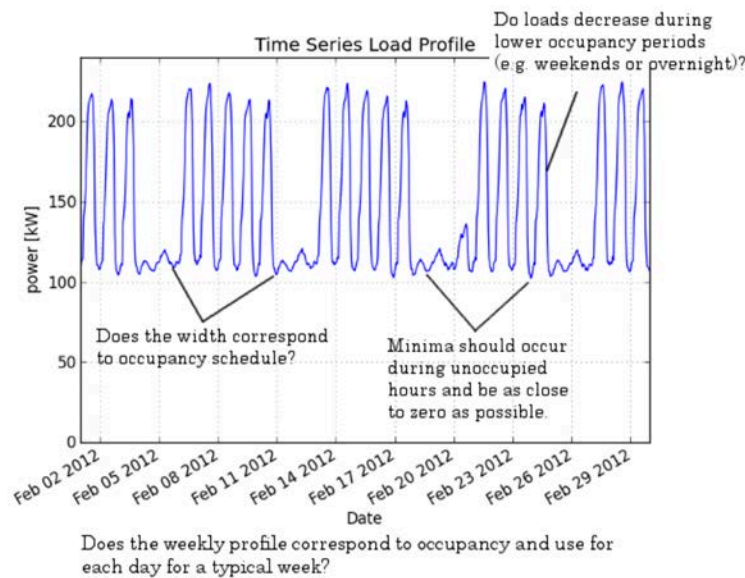


Figure 1. OpenEIS reference code output for time series load profiling

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 to 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.

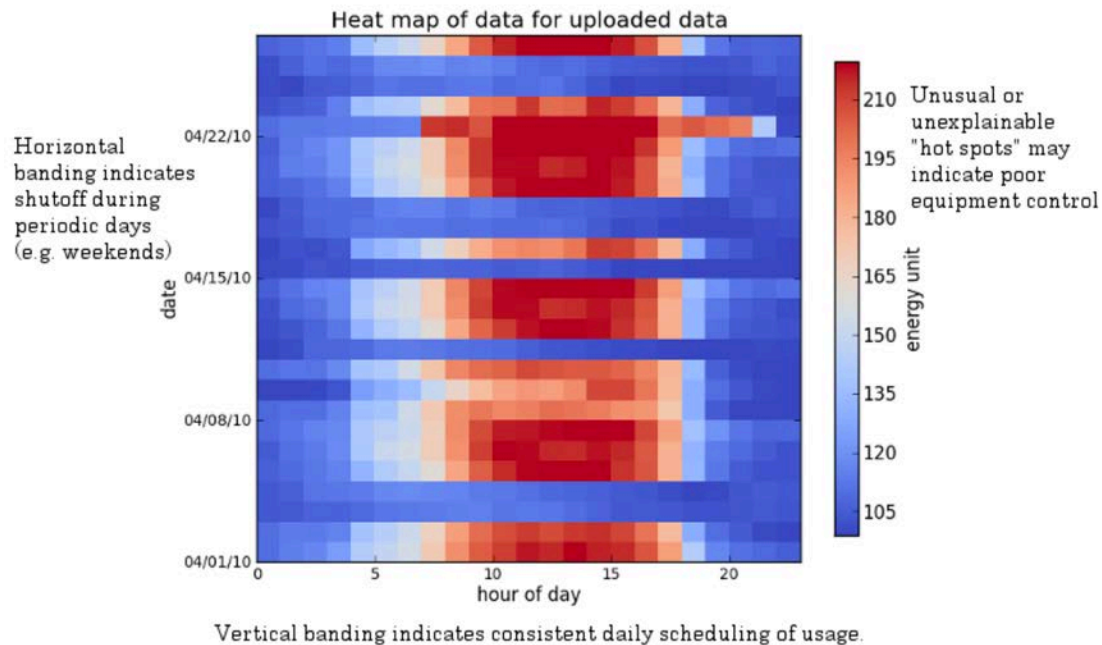


Figure 2. OpenEIS reference code output for heat maps

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.”

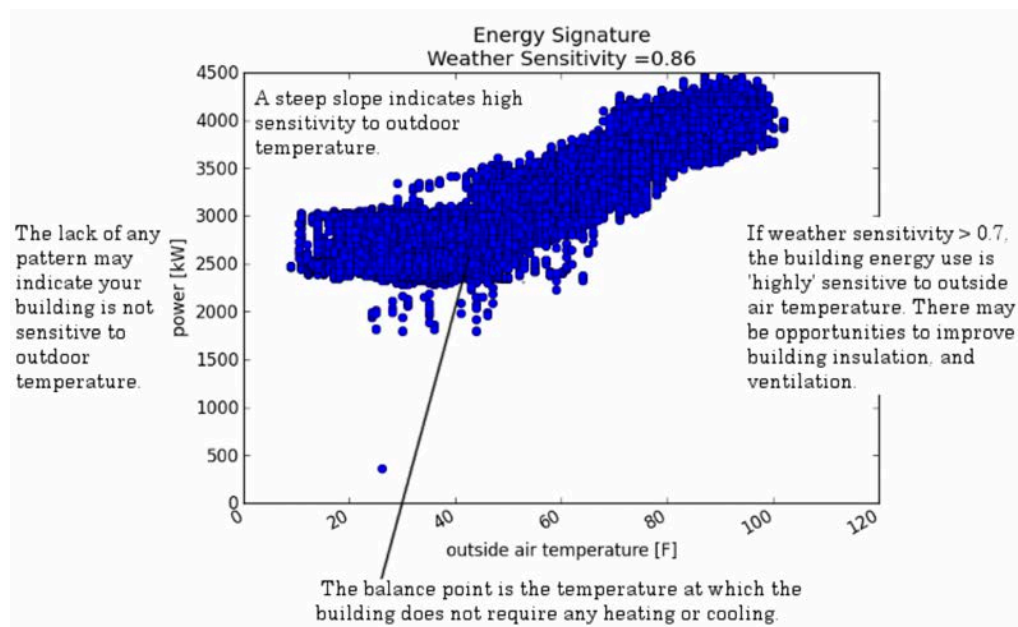


Figure 3. OpenEIS reference code output for energy signature

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:

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

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 subsets 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.

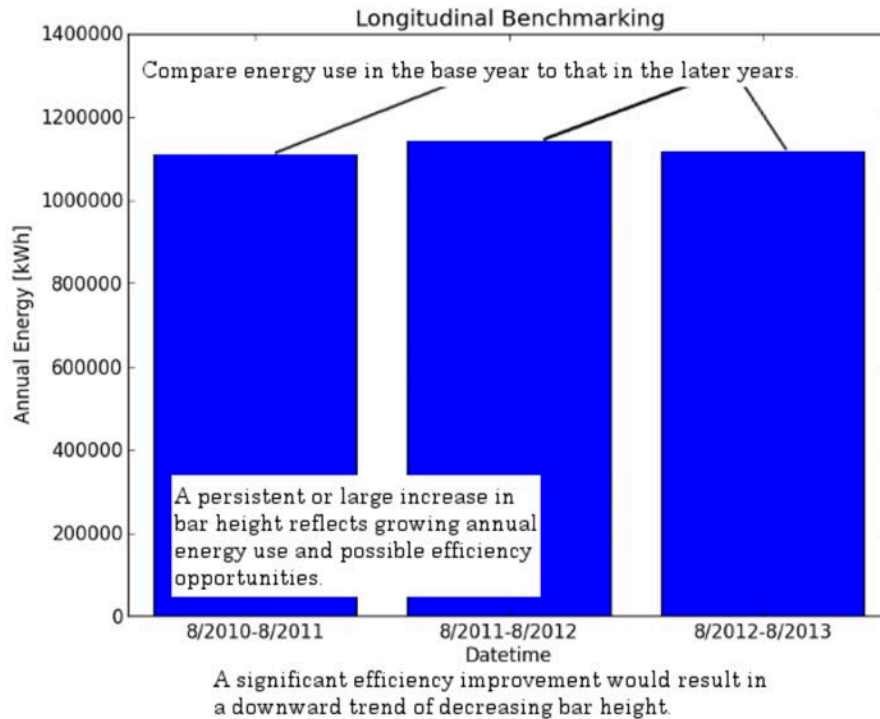
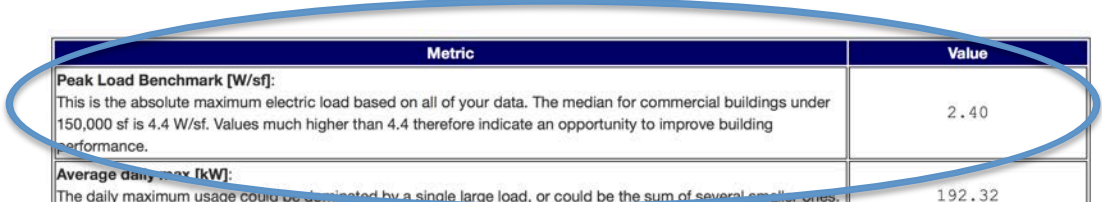


Figure 4. OpenEIS reference code output for longitudinal benchmarking

Longitudinal benchmarking is reviewed in detail in pages 51–58 in the *Energy Information Handbook*. In the OpenEIS reference code annual electricity consumption is represented in kWh, 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.

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.

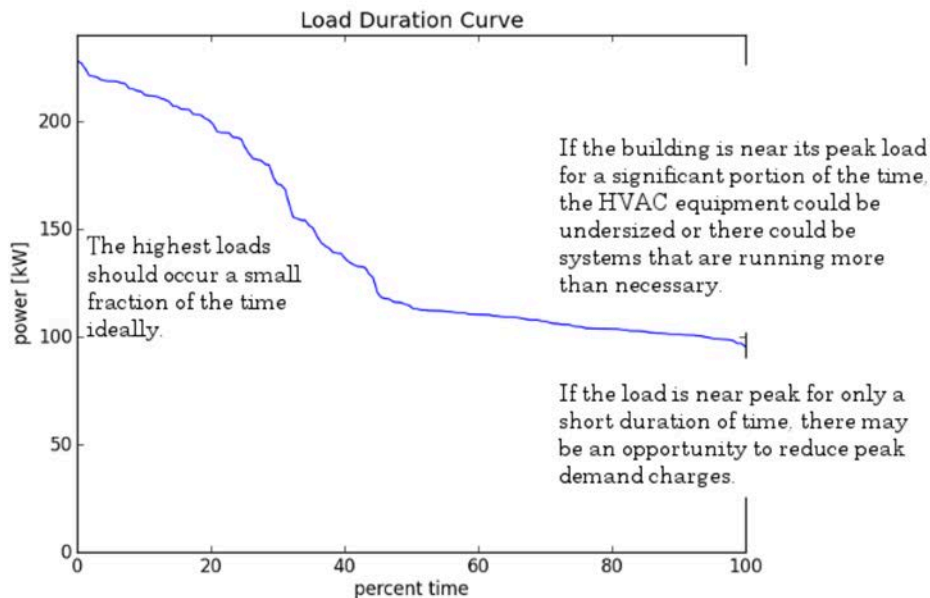


Figure 8. OpenEIS reference code output for load duration curve

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:

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

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.

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
Peak Load Benchmarking	Whole-building electric	Hourly
	Building floor area	n/a
Base-to-Peak Load Ratios	Whole-building electric	Hourly
Load Duration Curves	Whole-building electric	
Load Variability	Whole-building electric	

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 electricity, gas, and outside air temperature data.

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].

An example of a correctly formatted main data file is shown below in Figure 14.


```
Bldg90_1Week.csv - Notepad
File Edit Format View Help
Date,Hillside OAT [F],B-90 Main Meter - Trailers Subtracted [kw],*Boiler Gas Power Baseline (kBtu/hr)
6/1/2012 0:00,56.86,103.57,64.07
6/1/2012 1:00,55.6,107.28,89.03
6/1/2012 2:00,55.55,109.42,149.29
6/1/2012 3:00,55.87,110.89,274.23
6/1/2012 4:00,56.08,136.14,474.79
6/1/2012 5:00,57.3,141.65,734.73
6/1/2012 6:00,54.8,147.39,876.15
6/1/2012 7:00,54.69,159.77,767.89
6/1/2012 8:00,59.45,182.81,591.25
6/1/2012 9:00,61.36,199.16,423.62
6/1/2012 10:00,63.9,208.78,289.64
6/1/2012 11:00,69.64,210.04,260.18
6/1/2012 12:00,83.06,209.55,203.02
6/1/2012 13:00,87.55,219.07,121.04
6/1/2012 14:00,91.27,226.64,69.37
6/1/2012 15:00,91.77,224.26,57.79
6/1/2012 16:00,90.52,214.18,59.64
6/1/2012 17:00,88.21,192.65,55.78
6/1/2012 18:00,79.9,128.66,47.5
6/1/2012 19:00,65.59,108.96,30.28
6/1/2012 20:00,60.41,102.9,13.35
6/1/2012 21:00,58.36,102.45,0.16
6/1/2012 22:00,57.92,100.62,6.05
6/1/2012 23:00,55.74,100.1,52.24
6/2/2012 0:00,54.38,101.27,14.21
6/2/2012 1:00,54.14,102.15,25.98
6/2/2012 2:00,53.49,104.84,37.83
6/2/2012 3:00,53.23,104.98,43.17
6/2/2012 4:00,53.1,107.68,68.55
6/2/2012 5:00,52.79,107.79,87.89
6/2/2012 6:00,52.87,110.48,64.94
6/2/2012 7:00,54.18,112.16,53.09
6/2/2012 8:00,56.43,111.82,76.52
6/2/2012 9:00,58.68,112.02,94.24
6/2/2012 10:00,60.73,110.47,83.35
6/2/2012 11:00,66.46,110.45,52.68
6/2/2012 12:00,80.82,110.11,28.63
6/2/2012 13:00,84.83,109.73,25.14
6/2/2012 14:00,82.38,107.03,10.87
6/2/2012 15:00,83.15,106.86,4.43
6/2/2012 16:00,81.98,104.35,0.26
6/2/2012 17:00,79.66,102.45,0
```

Figure 14. Example of the main data file

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.

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”.

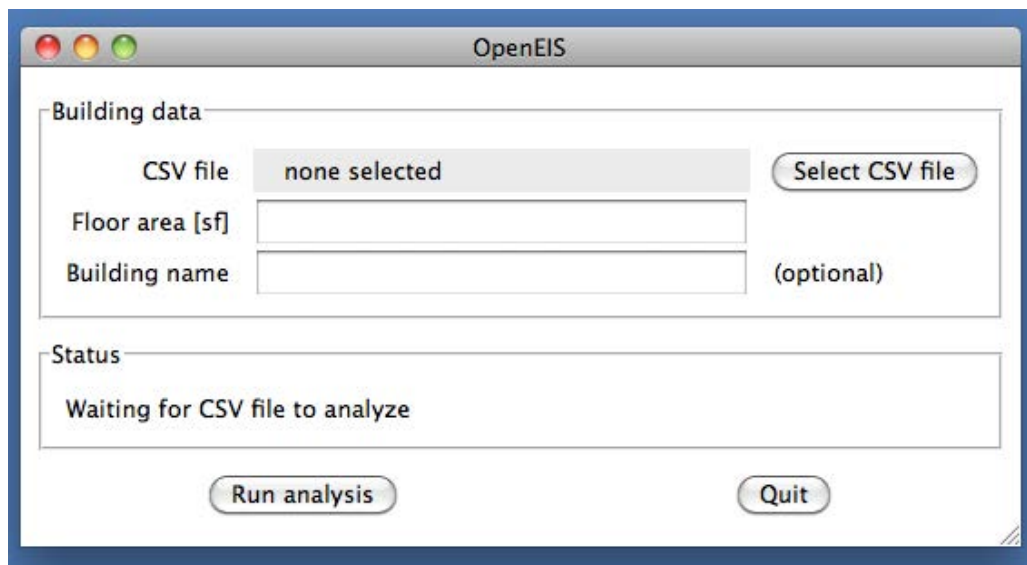
While the directory containing the reference suite can be relocated and renamed at will, this is not true of the “open_eis_lib” subdirectory. This subdirectory contains the scripts that implement the algorithms. Changing its name, the names of any of the files it contains, or the paths to any of the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

Code Execution

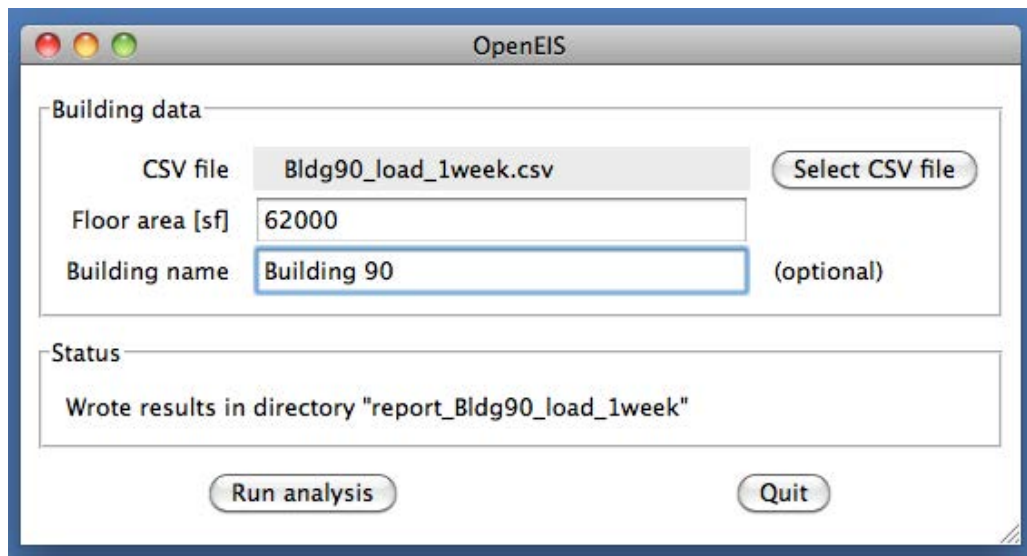
Next, run the algorithms on the sample data provided. To do so:

- 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:



- 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).
- 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:



- You may select another file, or choose “Quit” to close the window.

The output from running the algorithms is an html report located in the code directory. Open this file 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 $y = \text{Loads}$ as a function of $x = \text{oats}$.

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 [ft²] (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:
$$\text{spearmanCoeff} = \text{cosineFactor} / \sqrt{\text{xMagSq} * \text{yMagSq}}$$

- Note that the Spearman coefficient is the Pearson coefficient of the rank vectors *xRanks* and *yRanks*.
- Note that a correct calculation yields $-1 \leq \text{spearmanCoeff} \leq 1$.

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 *ii*th 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 [ft²] (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.

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.