thermostat Documentation

Release 1.0.0

Open Energy Efficiency, Inc.

November 23, 2016

Contents

This package and the savings calculation methods implemented herein are being developed in association with the EPA and industry stakeholders to help standardize calculations of temperature/run-time savings due to connected thermostats.

CHAPTER 1

Usage

1.1 Quickstart

1.1.1 Installation

To install the thermostat package for the first time, we highly recommend that you create a virtual environment or a conda environment in which to install it. You may choose to skip this step, but do so at the risk of corrupting your existing python environment. Isolating your python environment will also make it easier to debug.

```
# if using virtualenvwrapper (see https://virtualenvwrapper.readthedocs.org/en/latest/install.html)
$ mkvirtualenv thermostat
(thermostat)$ pip install thermostat

# if using conda (see note below - conda is distributed with Anaconda)
$ conda create --yes --name thermostat pandas
(thermostat)$ pip install thermostat
```

If you already have an environment, use the following:

```
# if using virtualenvwrapper
$ workon thermostat
(thermostat)$

# if using conda
$ source activate thermostat
(thermostat)$
```

To deactivate the environment when you've finished, use the following:

```
# if using virtualenvwrapper
(thermostat)$ deactivate
$

# if using conda
(thermostat)$ source deactivate
$
```

Check to make sure you are on the most recent version of the package.

```
>>> import thermostat; thermostat.get_version()
'1.0.0'
```

If you are not on the correct version, you should upgrade:

```
$ pip install thermostat --upgrade
```

The command above will update dependencies as well. If you wish to skip this, use the `--no-deps` flag:

```
$ pip install thermostat --upgrade --no-deps
```

Previous versions of the package are available on github.

Note: If you experience issues installing python packages with C extensions, such as *numpy* or *scipy*, we recommend installing and using the free Anaconda Python distribution by Continuum Analytics. It contains many of the numeric and scientific packages used by this package and has installers for Python 2.7 and 3.5 for Windows, Mac OS X and Linux.

Once you have verified a correct installation, import the necessary methods and set a directory for finding and storing data.

Note: If you suspect a package version conflict or error, you can verify the versions of the packages you have installed against the package versions in `thermostatreqnotes.txt`.

To list your package versions, use:

```
$ pip freeze
```

or (if you're using Anaconda):

```
$ conda list
```

1.1.2 Script setup and imports

Import the few built-in python packages and methods we will be using in this tutorial as follows.

```
import sys
import os
import warnings
from os.path import expanduser
```

Also make sure to import the methods we will be using from the thermostat package.

```
from thermostat.importers import from_csv
from thermostat.exporters import metrics_to_csv
from thermostat.stats import compute_summary_statistics
from thermostat.stats import summary_statistics_to_csv
```

Set the data_dir variable as a convenience. We will refer to this directory and save our results in it. You should also move all downloaded and extracted files used in this tutorial into this directory before using them. You may, of course, choose to use a different directory, which you can set here, or override it entirely by replacing it where it appears in the tutorial.

```
data_dir = os.path.join(expanduser("~"), "thermostat_tutorial")
# or data_dir = "/full/path/to/custom/directory/"
```

1.1.3 Optional Setup

If you wish to follow the progress of downloading and caching external weather files, which will be the most time-consuming portion of this tutorial, you may wish at this point to configure logging. The example here will work within most ipython or script environments. If you have a more complicated logging setup, you may need to use something other than the root logger, which this uses.

```
import logging
logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
```

Note: The thermostat package depends on the eemeter package for weather data fetching. The eemeter package automatically creates its own cache directory in which it keeps cached versions of weather source data. This speeds up the (generally I/O bound) NOAA weather fetching routine on subsequent internal calls to fetch the same weather data (i.e. getting outdoor temperature data for thermostats that map to the same weather station).

For more information, see the eemeter package.

Note: US Census Bureau ZIP Code Tabulation Areas (ZCTA) are used to map USPS ZIP codes to outdoor temperature data. If the automatic mapping is unsuccessful for one or more of the ZIP codes in your dataset, the reason is likely to be the discrepancy between "true" USPS ZIP codes and the US Census Bureau ZCTAs. "True" ZIP codes are not used because they do not always map well to location (for example, ZIP codes for P.O. boxes). You may need to first map ZIP codes to ZCTAs, or these thermostats will be skipped. There are roughly 32,000 ZCTAs and roughly 42000 ZIP codes - many fewer ZCTAs than ZIP codes.

1.1.4 Computing individual thermostat-season metrics

After importing the package methods, load the example thermostat data, or provide data of your own. See *Input data* for more detailed file format information.

Fabricated example data from 35 thermostats in various climate zones, is available for download `here`.

Loading the thermostat data below will take more than a few minutes, even if the weather cache is enabled (see note above). This is because loading thermostat data involves downloading hourly weather data from a remote source - in this case, the NCDC.

The following loads an lazy iterator over the thermostats. The thermostats will be loaded into memory as necessary in the following steps.

```
metadata_filename = os.path.join(data_dir, "examples/metadata.csv")
thermostats = from_csv(metadata_filename, verbose=True)
```

To calculate savings metrics, iterate through thermostats and save the results. Uncomment the commented lines if you would like to store the thermostats in memory for inspection. Note that this could eat up your application memory and is only recommended for debugging purposes.

```
metrics = []
# saved_thermostats = []
for thermostat in thermostats:
    outputs = thermostat.calculate_epa_field_savings_metrics()
    metrics.extend(outputs)
    # saved_thermostats.append(thermostat)
```

The single-thermostat metrics should be output to CSV and converted to dataframe format.

```
output_filename = os.path.join(data_dir, "thermostat_example_output.csv")
metrics_df = metrics_to_csv(metrics, output_filename)
```

The output CSV will be saved in your data directory and should very nearly match the output CSV provided in the example data.

See *Output data* for more detailed file format information.

1.1.5 Computing summary statistics

Once you have obtained output for each individual thermostat in your dataset, use the stats module to compute summary statistics, which are formatted for submission to the EPA. The example below works with the output file from the tutorial above and can be modified to use your data.

Compute statistics across all thermostats.

```
# uses the metrics_df created in the Quickstart above.
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    # uses the metrics_df created in the quickstart above.
    stats = compute_summary_statistics(metrics_df)

    # If you want to have advanced filter outputs, use this instead
    # stats_advanced = compute_summary_statistics(metrics_df, advanced_filtering=True)
```

Save these results to file.

Each row of the saved CSV will represent one type of output, with one row per statistic per output. Each column in the CSV will represent one subset of thermostats, as determined by grouping by EIC climate zone and applying various filtering methods. National weighted averages will be available near the top of the file.

At this point, you will also need to provide an alphanumeric product identifier for the connected thermostat; e.g. a combination of the connected thermostat service plus one or more connected thermostat device models that comprises the data set.

```
product_id = "INSERT ALPHANUMERIC PRODUCT ID HERE"
stats_filepath = os.path.join(data_dir, "thermostat_example_stats.csv")
stats_df = summary_statistics_to_csv(stats, stats_filepath, product_id)

# or with advanced filter outputs
# stats_advanced_filepath = os.path.join(data_dir, "thermostat_example_stats_advanced.csv")
# stats_advanced_df = summary_statistics_to_csv(stats_advanced, stats_advanced_filepath, product_id)
```

National savings are computed by weighted average of percent savings results grouped by climate zone. Heavier weights are applied to results in climate zones which, regionally, tend to have longer runtimes. Weightings used are available `for download`.

1.1.6 More information

For additional information on package usage, please see the *API* documentation.

1.1.7 Input data

Input data should be specified using the following formats. One CSV should specify thermostat summary metadata (e.g. unique identifiers, location, etc.). Another CSV (or CSVs) should contain runtime information, linked to the

metadata csv by the `thermostat_id` column.

Example files `here`.

Thermostat Summary Metadata CSV format

Columns

Name	Data Format	Units	Description
`thermostat_id`	string	N/A	A uniquely identifying marker for the thermostat.
`equipment_type`	enum, {0..5}	N/A	The type of controlled HVAC heating and cooling equipment. [1]
`zipcode`	string, 5 digits	N/A	The ZIP code in which the thermostat is installed [2].
`utc_offset`	string	N/A	The UTC offset of the times in the corresponding interval data CSV. (e.g. "-0700")
`interval_data_filename`	string	N/A	The filename of the interval data file corresponding to this thermostat. Should be specified relative to the location of the metadata file.

- Each row should correspond to a single thermostat.
- Nulls should be specified by leaving the field blank.
- All interval data for a particular thermostat should use the *same, single* UTC offset provided in the metadata file.

[1] Options for `equipment_type`:

- `0`: Other – e.g. multi-zone multi-stage, modulating. Note: module will not output savings data for this type.
- `1`: Single stage heat pump with electric resistance aux and/or emergency heat (i.e., strip heat)
- `2`: Single stage heat pump without additional and/or supplemental heating sources (excludes aux/emergency heat as well as dual fuel systems, i.e., heat pump plus gas- or oil-fired furnace)
- `3`: Single stage non heat pump with single-stage central air conditioning
- `4`: Single stage non heat pump without central air conditioning
- `5`: Single stage central air conditioning without central heating

[2] Will be used for matching with a weather station that provides external dry-bulb temperature data. This temperature data will be used to determine the bounds of the heating and cooling season over which metrics will be computed. For more information on the mapping between ZIP codes and weather stations, please see eemeter.weather.location.

Thermostat Interval Data CSV format

Columns

Name	Data Format	Units	Description
`thermostat_id`	string	N/A	Uniquely identifying marker for the thermostat.
`date`	YYYY-MM-DD (ISO-8601)	N/A	Date of this set of readings.
`cool_runtime`	decimal or integer	minutes	Daily runtime of cooling equipment.
`heat_runtime`	decimal or integer	minutes	Daily runtime of heating equipment. [3]
`auxiliary_heat_HH`	decimal or integer	minutes	Hourly runtime of auxiliary heat equipment (HH=00-23).
`emergency_heat_HH`	decimal or integer	minutes	Hourly runtime of emergency heat equipment (HH=00-23).
`temp_in_HH`	decimal, to nearest 0.5	°F	Hourly average conditioned space temperature over the period of the reading (HH=00-23).
`heating_setpoint_HH`	decimal, to nearest 0.5	°F	Hourly average thermostat setpoint temperature over the period of the reading (HH=00-23).
`cooling_setpoint_HH`	decimal, to nearest 0.5	°F	Hourly average thermostat setpoint temperature over the period of the reading (HH=00-23).

- Each row should correspond to a single daily reading from a thermostat.
- Nulls should be specified by leaving the field blank.
- Zero values should be specified as 0, rather than as blank.
- If data is missing for a particular row of one column, data should still be provided for other columns in that row. For example, if runtime is missing for a particular date, please still provide indoor conditioned space temperature and setpoints for that date, if available.
- Runtimes should be less than or equal to 1440 min (1 day).
- Dates should be specified in the ISO 8601 date format (e.g. `2015-05-19`).
- All temperatures should be specified in °F (to the nearest 0.5°F).
- If no distinction is made between heating and cooling setpoint, set both equal to the single setpoint.
- All runtime data MUST have the same UTC offset, as provided in the corresponding metadata file.
- If only a single setpoint is used for the thermostat, please copy the same setpoint data in to the heating and cooling setpoint columns.
- Outdoor temperature data need not be provided - it will be fetched automatically from NCDC using the eemeter package package.
- Dates should be consecutive.

1.1.8 Output data

Individual thermostat-season

The following columns are a intermediate output generated for each thermostat-season.

[3] Should not include runtime for auxiliary or emergency heat - this should be provided separately in the columns *emergency_heat_HH* and *auxiliary_heat_HH*.

Columns

Name	Data Format	Units	Des
General outputs			
sw_version	string	N/A	Soft
ct_identifier	string	N/A	Iden
equipment_type	enum {0..5}	N/A	Equi
heating_or_cooling	string	N/A	Labe
zipcode	string, 5 digits	N/A	ZIP
station	string, USAF ID	N/A	USA
climate_zone	string	N/A	EIC
start_date	date	ISO-8601	Earli
end_zone	date	ISO-8601	Late
n_days_both_heating_and_cooling	integer	# days	Num
n_days_insufficient_data	integer	# days	Num
n_core_cooling_days	integer	# days	Num
n_core_heating_days	integer	# days	Num
n_days_in_inputfile_date_range	integer	# days	Num
baseline10_core_cooling_comfort_temperature	float	°F	Base
baseline90_core_cooling_comfort_temperature	float	°F	Base
regional_average_baseline_cooling_comfort_temperature	float	°F	Base
regional_average_baseline_heating_comfort_temperature	float	°F	Base
Model outputs			
percent_savings_baseline_percentile	float	percent	Perc
avoided_daily_mean_core_day_runtime_baseline_percentile	float	minutes	Avoi
avoided_total_core_day_runtime_baseline_percentile	float	minutes	Avoi
baseline_daily_mean_core_day_runtime_baseline_percentile	float	minutes	Base
baseline_total_core_day_runtime_baseline_percentile	float	minutes	Base
percent_savings_baseline_regional	float	percent	Perc
avoided_daily_mean_core_day_runtime_baseline_regional	float	minutes	Avoi
avoided_total_core_day_runtime_baseline_regional	float	minutes	Avoi
baseline_daily_mean_core_day_runtime_baseline_regional	float	minutes	Base
baseline_total_core_day_runtime_baseline_regional	float	minutes	Base
mean_demand	float	°F	Aver
alpha	float	minutes/Δ°F	The
tau	float	°F	The
mean_sq_err	float	N/A	Mea
root_mean_sq_err	float	N/A	Root
cv_root_mean_sq_err	float	N/A	Coef
mean_abs_err	float	N/A	Mea
mean_abs_pct_err	float	N/A	Mea
Runtime outputs			
total_core_cooling_runtime	float	minutes	Tota
total_core_heating_runtime	float	minutes	Tota
total_auxiliary_heating_core_day_runtime	float	minutes	Tota
total_emergency_heating_core_day_runtime	float	minutes	Tota
daily_mean_core_cooling_runtime	float	minutes	Aver
daily_mean_core_heating_runtime	float	minutes	Aver
Resistance heat outputs			
rhu_00F_to_05F	decmial	0.0=0%, 1.0=100%	Resi

Table 1.1 – continued from previc

Name	Data Format	Units	Des
rhu_05F_to_10F	decmial	0.0=0%, 1.0=100%	Resi
rhu_10F_to_15F	decmial	0.0=0%, 1.0=100%	Resi
rhu_15F_to_20F	decmial	0.0=0%, 1.0=100%	Resi
rhu_20F_to_25F	decmial	0.0=0%, 1.0=100%	Resi
rhu_25F_to_30F	decmial	0.0=0%, 1.0=100%	Resi
rhu_30F_to_35F	decmial	0.0=0%, 1.0=100%	Resi
rhu_35F_to_40F	decmial	0.0=0%, 1.0=100%	Resi
rhu_40F_to_45F	decmial	0.0=0%, 1.0=100%	Resi
rhu_45F_to_50F	decmial	0.0=0%, 1.0=100%	Resi
rhu_50F_to_55F	decmial	0.0=0%, 1.0=100%	Resi
rhu_55F_to_60F	decmial	0.0=0%, 1.0=100%	Resi

Summary Statistics

For each real- or integer-valued column ("###") from the individual thermostat-season output, the following summary statistics are generated.

(For readability, these columns are actually rows.)

Columns

Name	Description
###_n	Number of samples
###_upper_bound_95_perc_conf	95% confidence upper bound on mean value
###_mean	Mean value
###_lower_bound_95_perc_conf	95% confidence lower bound on mean value
###_sem	Standard error of the mean
###_10q	1st decile (10th percentile, q=quantile)
###_20q	2nd decile
###_30q	3rd decile
###_40q	4th decile
###_50q	5th decile
###_60q	6th decile
###_70q	7th decile
###_80q	8th decile
###_90q	9th decile

The following general columns are also output:

Columns

Name	Description
sw_version	Software version
product_id	Alphanumeric product identifier
n_thermostat_core_day_sets_total	Number of relevant rows from thermostat module output before filtering
n_thermostat_core_day_sets_kept	Number of relevant rows from thermostat module not filtered out
n_thermostat_core_day_sets_discarded	Number of relevant rows from thermostat module filtered out

The following national weighted percent savings columns are also available.

National savings are computed by weighted average of percent savings results grouped by climate zone. Heavier weights are applied to results in climate zones which, regionally, tend to have longer runtimes. Weightings used are available `for download`.

Columns

Name	Description
percent_savings_baseline_percentile_mean	National weighted mean percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q10	National weighted 10th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q20	National weighted 20th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q30	National weighted 30th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q40	National weighted 40th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q50	National weighted 50th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q60	National weighted 60th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q70	National weighted 70th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q80	National weighted 80th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_q90	National weighted 90th percentile percent savings as given by baseline_percentile method.
percent_savings_baseline_percentile_lower_bound_mean	National weighted mean percent savings lower bound as given by a 95% confidence interval and the baseline_percentile method.
percent_savings_baseline_percentile_upper_bound_mean	National weighted mean percent savings upper bound as given by a 95% confidence interval and the baseline_percentile method.
percent_savings_baseline_regional_mean	National weighted mean percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q10	National weighted 10th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q20	National weighted 20th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q30	National weighted 30th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q40	National weighted 40th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q50	National weighted 50th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q60	National weighted 60th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q70	National weighted 70th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q80	National weighted 80th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_q90	National weighted 90th percentile percent savings as given by baseline_regional method.
percent_savings_baseline_regional_lower_bound_mean	National weighted mean percent savings lower bound as given by a 95% confidence interval and the baseline_regional method.
percent_savings_baseline_regional_upper_bound_mean	National weighted mean percent savings upper bound as given by a 95% confidence interval and the baseline_regional method.

1.2 API

1.2.1 thermostat.importers

thermostat.importers.**from_csv**(*metadata_filename*, *verbose=False*)

Creates Thermostat objects from data stored in CSV files.

Parameters

- **metadata_filename** (*str*) – Path to a file containing the thermostat metadata.
- **verbose** (*boolean*) – Set to True to output a more detailed log of import activity.

Returns **thermostats** – Thermostats imported from the given CSV input files.

Return type iterator over thermostat.Thermostat objects

thermostat.importers.**get_single_thermostat**(*thermostat_id*, *zipcode*, *equipment_type*, *utc_offset*, *interval_data_filename*)

Load a single thermostat directly from an interval data file.

Parameters

- **thermostat_id** (*str*) – A unique identifier for the thermostat.
- **zipcode** (*str*) – The zipcode of the thermostat, e.g. *"01234"*.
- **equipment_type** (*str*) – The equipment type of the thermostat.
- **utc_offset** (*str*) – A string representing the UTC offset of the interval data, e.g. *"-0700"*. Could also be *"Z"* (UTC), or just *"+7"* (equivalent to *"+0700"*), or any other timezone format recognized by the library method dateutil.parser.parse.
- **interval_data_filename** (*str*) – The path to the CSV in which the interval data is stored.

Returns **thermostat** – The loaded thermostat object.

Return type thermostat.Thermostat

1.2.2 thermostat.exporters

thermostat.exporters.**metrics_to_csv**(*metrics*, *filepath*)

Writes metrics outputs to the file specified.

Parameters

- **metrics** (*list of dict*) – list of outputs from the function *thermostat.calculate_epa_draft_rccs_field_savings_metrics()*
- **filepath** (*str*) – filepath specification for location of output CSV file.

Returns **df** – DataFrame containing data output to CSV.

Return type pd.DataFrame

1.2.3 thermostat.core

class thermostat.core.**CoreDaySet**(*name*, *daily*, *hourly*, *start_date*, *end_date*)

Bases: tuple

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

__getstate__()
Exclude the OrderedDict from pickling

__repr__()
Return a nicely formatted representation string

daily
Alias for field number 1

end_date
Alias for field number 4

hourly
Alias for field number 2

name
Alias for field number 0

start_date
Alias for field number 3

class thermostat.core.**Thermostat**(*thermostat_id*, *equipment_type*, *zipcode*, *station*, *temperature_in*, *temperature_out*, *cooling_setpoint*, *heating_setpoint*, *cool_runtime*, *heat_runtime*, *auxiliary_heat_runtime*, *emergency_heat_runtime*)

Bases: object

Main thermostat data container. Each parameter which contains timeseries data should be a pandas.Series with a datetimeIndex, and that each index should be equivalent.

Parameters

- **thermostat_id** (*object*) – An identifier for the thermostat. Can be anything, but should be identifying (e.g., an ID provided by the manufacturer).
- **equipment_type** (*{ 0, 1, 2, 3, 4, 5 }*) –
  - 0: Other - e.g. multi-zone multi-stage, modulating. Note: module will not output savings data for this type.
  - 1: Single stage heat pump with aux and/or emergency heat
  - 2: Single stage heat pump without aux or emergency heat
  - 3: Single stage non heat pump with single-stage central air conditioning
  - 4: Single stage non heat pump without central air conditioning
  - 5: Single stage central air conditioning without central heating
- **zipcode** (*str*) – Installation ZIP code for the thermostat.
- **station** (*str*) – USAF identifier for weather station used to pull outdoor temperature data.
- **temperature_in** (*pandas.Series*) – Contains internal temperature data in degrees Fahrenheit (F), with resolution of at least 0.5F. Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. freq='H').
- **heating_setpoint** (*pandas.Series*) – Contains target temperature (setpoint) data in degrees Fahrenheit (F), with resolution of at least 0.5F used to control heating equipment. Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. freq='H').

- **cooling_setpoint** (*pandas.Series*) – Contains target temperature (setpoint) data in degrees Fahrenheit (F), with resolution of at least 0.5F used to control cooling equipment. Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. `freq='H'`).
- **temperature_out** (*pandas.Series*) – Contains outdoor temperature (setpoint) data as observed by a relevant weather station in degrees Fahrenheit (F), with resolution of at least 0.5F. Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. `freq='H'`).
- **cool_runtime** (*pandas.Series,*) – Daily runtimes for cooling equipment controlled by the thermostat, measured in minutes. No datapoint should exceed 1440 mins, which would indicate over a day of runtime (impossible). Should be indexed by a pandas.DatetimeIndex with daily frequency (i.e. `freq='D'`).
- **heat_runtime** (*pandas.Series,*) – Daily runtimes for heating equipment controlled by the thermostat, measured in minutes. No datapoint should exceed 1440 mins, which would indicate over a day of runtime (impossible). Should be indexed by a pandas.DatetimeIndex with daily frequency (i.e. `freq='D'`).
- **auxiliary_heat_runtime** (*pandas.Series,*) – Hourly runtimes for auxiliary heating equipment controlled by the thermostat, measured in minutes. Auxiliary heat runtime is counted when both resistance heating and the compressor are running (for heat pump systems). No datapoint should exceed 60 mins, which would indicate over a hour of runtime (impossible). Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. `freq='H'`).
- **energency_heat_runtime** (*pandas.Series,*) – Hourly runtimes for emergency heating equipment controlled by the thermostat, measured in minutes. Emergency heat runtime is counted when resistance heating is running when the compressor is not (for heat pump systems). No datapoint should exceed 60 mins, which would indicate over a hour of runtime (impossible). Should be indexed by a pandas.DatetimeIndex with hourly frequency (i.e. `freq='H'`).

calculate_epa_field_savings_metrics(*core_cooling_day_set_method='entire_dataset'*, *core_heating_day_set_method='entire_dataset'*, *climate_zone_mapping=None*)

Calculates metrics for connected thermostat savings as defined by the specification defined by the EPA Energy Star program and stakeholders.

Parameters

- **core_cooling_day_set_method** (*{"entire_dataset", "year_end_to_end"}, default: "entire_dataset"*) – Method by which to find core cooling day sets.
  - "entire_dataset": all core cooling days in dataset (days with >= 1 hour of cooling runtime and no heating runtime.
  - "year_end_to_end": groups all core cooling days (days with >= 1 hour of total cooling and no heating) from January 1 to December 31 into independent core cooling day sets.
- **core_heating_day_set_method** (*{"entire_dataset", "year_mid_to_mid"}, default: "entire_dataset"*) – Method by which to find core heating day sets.
  - "entire_dataset": all core heating days in dataset (days with >= 1 hour of heating runtime and no cooling runtime.
  - "year_mid_to_mid": groups all core heating days (days with >= 1 hour of total heating and no cooling) from July 1 to June 30 into independent core heating day sets.

- **climate_zone_mapping** (*filename, default: None*) – A mapping from climate zone to zipcode. If None is provided, uses default zipcode to climate zone mapping provided in tutorial.

  default mapping

Returns **metrics** – list of dictionaries of output metrics; one per set of core heating or cooling days.

Return type list

get_baseline_cooling_demand(*core_cooling_day_set*, *temp_baseline*, *tau*)

Calculate baseline cooling demand for a particular core cooling day set and fitted physical parameters.

daily CTD base$_d$ $= \frac{\sum_{i=1}^{24}[\tau_c - \text{hourly } \Delta T \text{ base cool}_{d.n}]_+}{24}$, where

hourly ΔT base cool$_{d.n}(^\circ F)$ = base heat$T_{d.n}$ − hourly outdoor$T_{d.n}$, and

d is the core cooling day; $(001, 002, 003...x)$,

n is the hour; $(01, 02, 03...24)$,

τ_c (cooling), determined earlier, is a constant that is part of the CT/home's thermal/HVAC cooling run time model, and

$[]_+$ indicates that the term is zero if its value would be negative.

Parameters

- **core_cooling_day_set** (thermostat.core.CoreDaySet) – Core cooling days over which to calculate baseline cooling demand.
- **temp_baseline** (*float*) – Baseline comfort temperature
- **tau** (*float, default: None*) – From fitted demand model.

Returns **baseline_cooling_demand** – A series containing baseline daily heating demand for the core cooling day set.

Return type pandas.Series

get_baseline_cooling_runtime(*baseline_cooling_demand*, *alpha*)

Calculate baseline cooling runtime given baseline cooling demand and fitted physical parameters.

$RT_{\text{base cool}}(\text{minutes}) = \alpha_c \cdot \text{daily CTD base}_d$

Parameters

- **baseline_cooling_demand** (*pandas.Series*) – A series containing estimated daily baseline cooling demand.
- **alpha** (*float*) – Slope of fitted line

Returns **baseline_cooling_runtime** – A series containing estimated daily baseline cooling runtime.

Return type pandas.Series

get_baseline_heating_demand(*core_heating_day_set*, *temp_baseline*, *tau*)

Calculate baseline heating demand for a particular core heating day set and fitted physical parameters.

daily HTD base$_d$ $= \frac{\sum_{i=1}^{24}[\text{hourly } \Delta T \text{ base heat}_{d.n} - \tau_h]_+}{24}$, where

hourly ΔT base heat$_{d.n}(^\circ F)$ = base cool$T_{d.n}$ − hourly outdoor$T_{d.n}$, and

d is the core heating day; $(001, 002, 003...x)$,

n is the hour; $(01, 02, 03...24)$,

τ_h (heating), determined earlier, is a constant that is part of the CT/home's thermal/HVAC heating run time model, and

$[]_+$ indicates that the term is zero if its value would be negative.

Parameters

- **core_heating_day_set** (thermostat.core.CoreDaySet) – Core heating days over which to calculate baseline cooling demand.
- **temp_baseline** (*float*) – Baseline comfort temperature
- **tau** (*float, default: None*) – From fitted demand model.

Returns **baseline_heating_demand** – A series containing baseline daily heating demand for the core heating days.

Return type pandas.Series

get_baseline_heating_runtime(*baseline_heating_demand*, *alpha*)

Calculate baseline heating runtime given baseline heating demand. and fitted physical parameters.

$RT_{\text{base heat}}(\text{minutes}) = \alpha_h \cdot \text{daily HTD base}_d$

Parameters

- **baseline_heating_demand** (*pandas.Series*) – A series containing estimated daily baseline heating demand.
- **alpha** (*float*) – Slope of fitted line

Returns **baseline_heating_runtime** – A series containing estimated daily baseline heating runtime.

Return type pandas.Series

get_cooling_demand(*core_cooling_day_set*)

Calculates a measure of cooling demand using the hourlyavgCTD method.

Starting with an assumed value of zero for Tau (τ_c), calculate the daily Cooling Thermal Demand (daily CTD_d), as follows

$\text{daily CTD}_d = \frac{\sum_{i=1}^{24}[\tau_c - \text{hourly}\Delta T_{d.n}]_+}{24}$, where

$\text{hourly}\Delta T_{d.n}(^\circ F) = \text{hourly indoor}T_{d.n} - \text{hourly outdoor}T_{d.n}$, and

d is the core cooling day; $(001, 002, 003...x)$,

n is the hour; $(01, 02, 03...24)$,

τ_c (cooling) is the ΔT associated with $CTD = 0$ (zero cooling runtime), and

$[]_+$ indicates that the term is zero if its value would be negative.

For the set of all core cooling days in the CT interval data file, use ratio estimation to calculate α_c, the home's responsiveness to cooling, which should be positive.

$\alpha_c\left(\frac{\text{minutes}}{^\circ F}\right) = \frac{RT_{\text{actual cool}}}{\sum_{d=1}^{x}\text{daily CTD}_d}$, where

$RT_{\text{actual cool}}$ is the sum of cooling run times for all core cooling days in the CT interval data file.

For the set of all core cooling days in the CT interval data file, optimize τ_c that results in minimization of the sum of squares of the difference between daily run times reported by the CT, and calculated daily cooling run times.

Next recalculate α_c (in accordance with the above step) and record the model's parameters (α_c, τ_c)

Parameters **core_cooling_day_set** (thermostat.core.CoreDaySet) – Core day set over which to calculate cooling demand.

Returns

- **demand** (*pd.Series*) – Daily demand in the core heating day set as calculated using the method described above.
- **tau** (*float*) – Estimate of τ_c.
- **alpha** (*float*) – Estimate of α_c
- **mse** (*float*) – Mean squared error in runtime estimates.
- **rmse** (*float*) – Root mean squared error in runtime estimates.
- **cvrmse** (*float*) – Coefficient of variation of root mean squared error in runtime estimates.
- **mape** (*float*) – Mean absolute percent error
- **mae** (*float*) – Mean absolute error

get_core_cooling_day_baseline_setpoint (*core_cooling_day_set*, *method='tenth_percentile'*, *source='temperature_in'*)

Calculate the core cooling day baseline setpoint (comfort temperature).

Parameters

- **core_cooling_day_set** (*thermost.core.CoreDaySet*) – Core cooling days over which to calculate baseline cooling setpoint.
- **method** (*{"tenth_percentile"}, default: "tenth_percentile"*) – Method to use in calculation of the baseline.
  - "tenth_percentile": 10th percentile of source temperature. (Either cooling setpoint or temperature in).
- **source** (*{"cooling_setpoint", "temperature_in"}, default "temperature_in"*) – The source of temperatures to use in baseline calculation.

Returns **baseline** – The baseline cooling setpoint for the core cooling days as determined by the given method.

Return type float

get_core_cooling_days (*method='entire_dataset'*, *min_minutes_cooling=30*, *max_minutes_heating=0*)

Determine core cooling days from data associated with this thermostat.

Parameters

- **method** (*{"entire_dataset", "year_end_to_end"}, default: "entire_dataset"*) – Method by which to find core cooling days.
  - "entire_dataset": all cooling days in dataset (days with >= 30 min of cooling runtime and no heating runtime.
  - "year_end_to_end": groups all cooling days (days with >= 30 min of total cooling and no heating) from January 1 to December 31 into individual core cooling sets.
- **min_minutes_cooling** (*int, default 30*) – Number of minutes of core cooling runtime per day required for inclusion in core cooling day set.

- **max_minutes_heating** (*int, default 0*) – Number of minutes of heating runtime per day beyond which the day is considered part of a shoulder season (and is therefore not part of the core cooling day set).

Returns

core_cooling_day_sets – List of core day sets detected; Core day sets are represented as pandas Series of boolean values, intended to be used as selectors or masks on the thermostat data at hourly and daily frequencies.

A value of True at a particular index indicates inclusion of of the data at that index in the core day set. If method is "entire_dataset", name of core day set is "cooling_ALL"; if method is "year_end_to_end", names of core day sets are of the form "cooling_YYYY"

Return type list of thermostat.core.CoreDaySet objects

get_core_day_set_n_days(*core_day_set*)

Returns number of days in the core day set.

get_core_heating_day_baseline_setpoint(*core_heating_day_set*, *method='ninetieth_percentile'*, *source='temperature_in'*)

Calculate the core heating day baseline setpoint (comfort temperature).

Parameters

- **core_heating_day_set** (thermostat.core.CoreDaySet) – Core heating days over which to calculate baseline heating setpoint.
- **method** (*{"ninetieth_percentile"}, default: "ninetieth_percentile"*) – Method to use in calculation of the baseline.
  - "ninetieth_percentile": 90th percentile of source temperature. (Either heating setpoint or indoor temperature).
- **source** (*{"heating_setpoint", "temperature_in"}, default "temperature_in"*) – The source of temperatures to use in baseline calculation.

Returns **baseline** – The baseline heating setpoint for the heating day as determined by the given method.

Return type float

get_core_heating_days(*method='entire_dataset'*, *min_minutes_heating=30*, *max_minutes_cooling=0*)

Determine core heating days from data associated with this thermostat

Parameters

- **method** (*{"entire_dataset", "year_mid_to_mid"}, default: "entire_dataset"*) – Method by which to find core heating day sets.
  - "entire_dataset": all heating days in dataset (days with >= 30 min of heating runtime and no cooling runtime. (default)
  - "year_mid_to_mid": groups all heating days (days with >= 30 min of total heating and no cooling) from July 1 to June 30 (inclusive) into individual core heating day sets. May overlap with core cooling day sets.
- **min_minutes_heating** (*int, default 30*) – Number of minutes of heating runtime per day required for inclusion in core heating day set.

- **max_minutes_cooling** (*int, default 0*) – Number of minutes of cooling runtime per day beyond which the day is considered part of a shoulder season (and is therefore not part of the core heating day set).

Returns

core_heating_day_sets – List of core day sets detected; Core day sets are represented as pandas Series of boolean values, intended to be used as selectors or masks on the thermostat data at hourly and daily frequencies.

A value of True at a particular index indicates inclusion of of the data at that index in the core day set. If method is "entire_dataset", name of core day sets are "heating_ALL"; if method is "year_mid_to_mid", names of core day sets are of the form "heating_YYYY-YYYY"

Return type list of thermostat.core.CoreDaySet objects

get_heating_demand(*core_heating_day_set*)

Calculates a measure of heating demand using the hourlyavgCTD method.

daily $\text{HTD}_d = \frac{\sum_{i=1}^{24}[\text{hourly}\Delta T_{d.n} - \tau_h]_+}{24}$, where

$\text{hourly}\Delta T_{d.n}(^\circ F) = \text{hourly indoor}T_{d.n} - \text{hourly outdoor}T_{d.n}$, and

d is the core heating day; $(001, 002, 003...x)$,

n is the hour; $(01, 02, 03...24)$,

τ_h (heating) is the ΔT associated with $HTD = 0$, reflecting that homes with no heat running tend to be warmer that the outdoors, and

$[]_+$ indicates that the term is zero if its value would be negative.

For the set of all core heating days in the CT interval data file, use ratio estimation to calculate α_h, the home's responsiveness to heating, which should be positive.

$\alpha_h \left(\frac{\text{minutes}}{^\circ F}\right) = \frac{RT_{\text{actual heat}}}{\sum_{d=1}^{x} \text{daily HTD}_d}$, where

$RT_{\text{actual heat}}$ is the sum of heating run times for all core heating days in the CT interval data file.

For the set of all core heating days in the CT interval data file, optimize τ_h that results in minimization of the sum of squares of the difference between daily run times reported by the CT, and calculated daily heating run times.

Next recalculate α_h (in accordance with the above step) and record the model's parameters (α_h, τ_h)

Parameters **core_heating_day_set** (*array_like*) – Core day set over which to calculate heating demand.

Returns

- **demand** (*pd.Series*) – Daily demand in the core heating day set as calculated using the method described above.
- **tau** (*float*) – Estimate of τ_h.
- **alpha** (*float*) – Estimate of α_h
- **mse** (*float*) – Mean squared error in runtime estimates.
- **rmse** (*float*) – Root mean squared error in runtime estimates.
- **cvrmse** (*float*) – Coefficient of variation of root mean squared error in runtime estimates.
- **mape** (*float*) – Mean absolute percent error
- **mae** (*float*) – Mean absolute error

get_ignored_days (*core_day_set*)
Determine how many days are ignored for a particular core day set

Returns

- **n_both** (*int*) – Number of days excluded from core day set because of presence of both heating and cooling runtime.
- **n_days_insufficient** (*int*) – Number of days excluded from core day set because of null runtime data.

get_inputfile_date_range (*core_day_set*)
Returns number of days of data provided in input data file.

get_resistance_heat_utilization_bins (*core_heating_day_set*)
Calculates resistance heat utilization metrics in temperature bins of 5 degrees between 0 and 60 degrees Fahrenheit.

Parameters **core_heating_day_set** (thermostat.core.CoreDaySet) – Core heating day set for which to calculate total runtime.

Returns **RHUs** – Resistance heat utilization for each temperature bin, ordered ascending by temperature bin. Returns None if the thermostat does not control the appropriate equipment

Return type numpy.array or None

total_auxiliary_heating_runtime (*core_day_set*)
Calculates total auxiliary heating runtime.

Parameters **core_day_set** (thermostat.core.CoreDaySet) – Core day set for which to calculate total runtime.

Returns **total_runtime** – Total auxiliary heating runtime.

Return type float

total_cooling_runtime (*core_day_set*)
Calculates total cooling runtime.

Parameters **core_day_set** (thermostat.core.CoreDaySet) – Core day set for which to calculate total runtime.

Returns **total_runtime** – Total cooling runtime.

Return type float

total_emergency_heating_runtime (*core_day_set*)
Calculates total emergency heating runtime.

Parameters **core_day_set** (thermostat.core.CoreDaySet) – Core day set for which to calculate total runtime.

Returns **total_runtime** – Total heating runtime.

Return type float

total_heating_runtime (*core_day_set*)
Calculates total heating runtime.

Parameters **core_day_set** (thermostat.core.CoreDaySet) – Core day set for which to calculate total runtime.

Returns **total_runtime** – Total heating runtime.

Return type float

1.2.4 thermostat.regression

thermostat.regression.**runtime_regression**(*daily_runtime*, *daily_demand*, *method*)

Least squares regession of runtime against a measure of demand.

Parameters

- **hourly_runtime** (*pd.Series with pd.DatetimeIndex*) – Runtimes for a particular heating or cooling season.
- **daily_demand** (*pd.Series with pd.DatetimeIndex*) – A daily demand measure for each day in the heating or cooling season.

Returns

- **slope** (*float*) – The slope parameter found by the regression to minimize sq error
- **intercept** (*float*) – The intercept parameter found by the regression to minimize sq error
- **mean_sq_err** (*float*) – The mean squared error of the regession.
- **root_mean_sq_err** (*float*) – The root mean squared error of the regession.

1.2.5 thermostat.stats

thermostat.stats.**combine_output_dataframes**(*dfs*)

Combines output dataframes. Useful when combining output from batches.

Parameters **dfs** (*list of pd.DataFrame*) – Output dataFrames to combine into one.

Returns **out** – Dataframe with combined output metadata.

Return type pd.DataFrame

thermostat.stats.**compute_summary_statistics**(*metrics_df*, *target_baseline_method='baseline_percentile'*, *advanced_filtering=False*)

Computes summary statistics for the output dataframe. Computes the following statistics for each real-valued or integer valued column in the output dataframe: mean, standard error of the mean, and deciles.

Parameters

- **df** (*pd.DataFrame*) – Output for which to compute summary statistics.
- **label** (*str*) – Name for this set of thermostat outputs.
- **target_baseline_method** (*{"baseline_percentile", "baseline_regional"}, default "baseline_percentile"*) – Baselining method by which samples will be filtered according to bad fits.

Returns

stats – An ordered dict containing the summary statistics. Column names are as follows, in which ### is a placeholder for the name of the column:

- mean: ###_mean
- standard error of the mean: ###_sem
- 10th quantile: ###_10q
- 20th quantile: ###_20q
- 30th quantile: ###_30q

- 40th quantile: ###_40q
- 50th quantile: ###_50q
- 60th quantile: ###_60q
- 70th quantile: ###_70q
- 80th quantile: ###_80q
- 90th quantile: ###_90q
- number of non-null core day sets: ###_n

The following general values are also output:

- label: label
- number of total core day sets: n_total_core_day_sets

Return type collections.OrderedDict

thermostat.stats.**get_filtered_stats**(*df*, *row_filter*, *label*, *heating_or_cooling*, *target_columns*, *target_baseline_method*)

thermostat.stats.**summary_statistics_to_csv**(*stats*, *filepath*, *product_id*)

Write metric statistics to CSV file.

Parameters

- **stats** (*list of dict*) – List of outputs from thermostat.stats.compute_summary_statistics()
- **filepath** (*str*) – Filepath at which to save the suppary statistics
- **product_id** (*str*) – A combination of the connected thermostat service plus one or more connected thermostat device models that comprises the data set.

Returns **df** – A pandas dataframe containing the output data.

Return type pandas.DataFrame

1.2.6 thermostat.parallel

thermostat.parallel.**schedule_batches**(*metadata_filename*, *n_batches*, *zip_files=False*, *batches_dir=None*)

Batch scheduler for large sets of thermostats. Can either create zipped directories ready be sent to separate processors for parallel processing, or unpackaged metadata dataframes for more flexible processing.

Parameters

- **metadata_filename** (*str*) – Full path to location of file containing CSV formatted metadata for
- **n_batches** (*int*) – Number of batches desired. Should be <= the number of available thermostats.
- **zip_files** (*boolean*) – If True, create zipped directories of metadata and interval data. Each batch will be named batch_XXXXX.zip, and will contain a directory named *data*, which contains metadata and interval data for the batch. Must supply *batches_dir* argument to use this option.
- **batches_dir** (*str*) – Path to directory in which to save created batches. Ignored for zip_files=False.

Returns batches – If zip_files is True, then returns list of names of created zip files. Otherwise, returns list of metadata dataframes containing batches.

Return type list of str or list of pd.DataFrame

CHAPTER 2

License

MIT

CHAPTER 3

Contact

Please feel free to reach out to either Dan Baldewicz (Dan.Baldewicz@icfi.com, 518-452-6426) or Phil Ngo (phil@theimpactlab.co) with questions or for technical support.

t

Symbols

C

D

E

F

G

H

M

N

R

S

T