Connected Thermostat Stakeholder Instructions for Data Filtering

July 1, 2016

Instructions:

EPA requests that the filtering tests are applied to each metric (DeltaT, Daily, Hourly), so we know if a filtering rule will have unequal impacts on different metrics. These filters are placed in the sample code in order, and commented for ease of use (apply/remove).

We hope that you will experiment with which filters to apply and the filtering levels on each metric to determine what has the optimal results for your data. Your data may, for example, indicate that filtering on CVRMSE did not impact the results, etc.

We would like your observations and commentary on the pros and cons of the filters in the context of each metric, what your recommendation are for which filters to apply to the DataCall and at what levels.

Also, we would like to analyze these results, so once you determine your recommended filter levels, please send us your recommended filter levels (for each metric) and the raw statistics module output/results for the Control (no filters), your filters applied to DeltaT, your filters applied to Daily, and your filters applied to Hourly.

Please send this information to Dan Baldewicz (dan.baldewicz@icfi.com) and Doug Frazee (douglas.frazee@icfi.com) by **July 18th**, so we can code these recommendations into the thermostat module, for use in Datacall 3 in Early August.

Filter Information:

The Filters, and recommended order are:

1. τ – place min and max acceptable limits for τ, i.e. number of degrees the inside temperature rises above outside temperature in the absence of HVAC run times (Physics Based, high or low τ imply poor representation of that home in the model)
2. Limit maximum CVRMSE (Regression fit quality).
3. Trim CTs with very low and/or very high savings by a percent of the total number of CTs in the pool

Feel free to adjust the parameters to match your insights based on your data.

A possible starting point:

1. $0°F \leq \tau \leq 20°F$
2. CVRMSE <= .4
3. Trim top and bottom 1%

Placement In Quickstart Program:

```
output_filename = os.path.join(data_dir, "thermostat_example_output2.csv")
metrics_df = seasonal_metrics_to_csv(seasonal_metrics, output_filename)

#FILTER A - FLOAT/ BALANCE TEMP (pandas?)    !!FUNDAMENTAL Physics           !!
#Filter B on RMSE                            !!HARD to Draw limits/sensitivy !!
#FILter C on TRIM - OUTLIER %                !!LAST CHANCE TO ELIM OUTLIER   !!
#FILTER D no negative/>100% savings          !!  ~                           !!

#NEW METRICS DF -->


from thermostat.stats import compute_summary_statistics
from thermostat.stats import summary_statistics_to_csv

from thermostat.stats import compute_summary_statistics_by_zipcode_group

stats = compute_summary_statistics(metrics_df, "all_thermostats")

stats_filepath = os.path.join(data_dir, "thermostat_example_stats1.csv")
stats_df = summary_statistics_to_csv(stats, stats_filepath)
```

INSERT FILTERING CODE HERE

Code:

```
# helper functions
def apply_filters(filters, df):
    """
    Applies a set of boolean series filters to filter the
    metrics_df and return a filtered copy.
    """
    filtered = df.copy()
    print("Number of rows before filtering: {:>6}".format(filtered.shape[0]))
    for filter_ in filters:
        filtered = filtered[filter_]
    print("Number of rows after filtering:  {:>6}".format(filtered.shape[0]))
    return filtered

def is_heating(name):
    return "Heating" in name

def is_cooling(name):
    return "Cooling" in name
```

```
# These filters are each boolean pandas Series, they are
# used to index the metrics dataframe. Note that these are
# positive filters - `True` values will be included,
# `False` values will be filtered out.
#
# Swap in any column of the metrics_df here.
#  For example:
#   metrics_df.ANYCOLUMN [>,<,<=,>=,==] ANYVALUE
#
# Note that the `... | metrics_df.season_name.map(is_heating)`
# pattern that appears in some filters restricts the filter
# to heating seasons or cooling seasons. The `|` operator is
# an elementwise boolean "or".
#
# The np.inf value represents "postive infinity" - setting
# this to the filter value (or -np.inf, for ">" filters)
# is equivalent to deactivating the filter.
#
# Outputs will be stored in the variable `metrics_df_filtered`,
# which does not overwrite metrics_df. You will need to edit
# your script to use `metrics_df_filtered` below the point of
# filtering.

filters = [

#example of filtering based on Float (Tau)
metrics_df.mean_demand_deltaT  <= 20.0
metrics_df.mean_demand_dailyavgCDD <= 20.0
metrics_df.mean_demand_dailyavgHDD <= 20.0
metrics_df.mean_demand_hourlyavgCDD <= 20.0
metrics_df.mean_demand_hourlyavgHDD <= 20.0

    # example of filtering cutoff (dailyavgCDD and dailyavgHDD)
    metrics_df.cv_root_mean_sq_err_deltaT < np.inf,
    (metrics_df.cv_root_mean_sq_err_dailyavgCDD < 0.4) |
metrics_df.season_name.map(is_heating),
    (metrics_df.cv_root_mean_sq_err_dailyavgHDD < 0.4) |
metrics_df.season_name.map(is_cooling),
    (metrics_df.cv_root_mean_sq_err_hourlyavgCDD < np.inf) |
metrics_df.season_name.map(is_heating),
    (metrics_df.cv_root_mean_sq_err_hourlyavgHDD < np.inf) |
metrics_df.season_name.map(is_cooling),

    # example of filtering within range (hourlyavgCDD and hourlyavgHDD)
    metrics df.percent savings deltaT < np.inf,
    (metrics_df.percent_savings_dailyavgCDD < np.inf) |
metrics_df.season_name.map(is_heating),
    (metrics_df.percent_savings_dailyavgHDD < np.inf) |
metrics_df.season_name.map(is_cooling),
    (metrics_df.percent_savings_hourlyavgCDD < 100) |
metrics_df.season_name.map(is_heating),
    (metrics_df.percent_savings_hourlyavgHDD < 100) |
metrics_df.season_name.map(is_cooling),

    metrics df.percent savings deltaT > -np.inf,
    (metrics_df.percent_savings_dailyavgCDD > -np.inf) |
metrics_df.season_name.map(is_heating),
    (metrics_df.percent_savings_dailyavgHDD > -np.inf) |
metrics_df.season_name.map(is_cooling),
    (metrics_df.percent_savings_hourlyavgCDD > -100) |
```

```
metrics_df.season_name.map(is_heating),
    (metrics_df.percent_savings_hourlyavgHDD > -100) |
metrics_df.season_name.map(is_cooling),

    # example of filtering out beyond 98th percentile and below the 2nd percentile
(deltaT)
    metrics_df.root_mean_sq_err_deltaT <
metrics_df.root_mean_sq_err_deltaT.dropna().quantile(0.98),
    (metrics_df.root_mean_sq_err_dailyavgCDD <
metrics_df.root_mean_sq_err_dailyavgCDD.dropna().quantile(1.0)) |
metrics_df.season_name.map(is_heating),
    (metrics_df.root_mean_sq_err_dailyavgHDD <
metrics_df.root_mean_sq_err_dailyavgHDD.dropna().quantile(1.0)) |
metrics_df.season_name.map(is_cooling),
    (metrics_df.root_mean_sq_err_hourlyavgCDD <
metrics_df.root_mean_sq_err_hourlyavgCDD.dropna().quantile(1.0)) |
metrics_df.season_name.map(is_heating),
    (metrics_df.root_mean_sq_err_hourlyavgHDD <
metrics_df.root_mean_sq_err_hourlyavgHDD.dropna().quantile(1.0)) |
metrics_df.season_name.map(is_cooling),

    metrics_df.root_mean_sq_err_deltaT >
metrics_df.root_mean_sq_err_deltaT.dropna().quantile(0.02),
    (metrics_df.root_mean_sq_err_dailyavgCDD >
metrics_df.root_mean_sq_err_dailyavgCDD.dropna().quantile(0.00)) |
metrics_df.season_name.map(is_heating),
    (metrics_df.root_mean_sq_err_dailyavgHDD >
metrics_df.root_mean_sq_err_dailyavgHDD.dropna().quantile(0.00)) |
metrics_df.season_name.map(is_cooling),
    (metrics_df.root_mean_sq_err_hourlyavgCDD >
metrics_df.root_mean_sq_err_hourlyavgCDD.dropna().quantile(0.00)) |
metrics_df.season_name.map(is_heating),
    (metrics_df.root_mean_sq_err_hourlyavgHDD >
metrics_df.root_mean_sq_err_hourlyavgHDD.dropna().quantile(0.00)) |
metrics_df.season_name.map(is_cooling),
]

metrics_df_filtered = apply_filters(filters, metrics_df)
```

Contact:

Primary:

Dan Baldewicz, ICF International

Phone: 518-452-6426

Email: Dan.Baldewicz@icfi.com

Secondary:

Phil Ngo, Impact Labs

Email: phil@theimpactlab.co