**EWX Flow Rules & Algorithms**

***Aggregation / Detection / Editing...***

***Estimation***

|  |  |
| --- | --- |
| Linear Interpolation  *number of points* | * Uses gap check, zero reads, and variance validation to fill missing or extreme data with linearly interpolated data * Decides which chunks need interpolation based on if they have gaps, low reads, or spikes/dips * Linearly interpolates if the chunks are at least ***number of points*** |
| Like Day Estimation  *number like days*  *max weeks to use* | * Fills gaps with average usage from like days using ***number of like days*** * Searches forwards or backwards up to ***max weeks to use*** to get like day usage data |

***Forecasting / Forwarding / Storage...***

***Validation***

|  |  |
| --- | --- |
| Interval Gap Check | * Checks for time gaps in interval usage data |
| Nonperiodic Zero Reads Check IDR  *margin*  *common threshold* | * Calculates proportion of reads less than (or equal to) ***margin***, grouped by weekday and hour * For each usage, if the proportion is less than ***common threshold***, these reads are nonperiodic. Otherwise, they are periodic – meaning it is ok for usage on this day and time to be zero ***common threshold*** percent of the time |
| Energy Sum Validation  *threshold* | * Checks periods based on heartbeats to determine if IDR sum and scalar values are similar * Flags end of period if   *sum(interval) =/= scalar +/-* ***threshold*** %   * Calculates percent diff b/t IDR sum & scalar as   *(IDR sum – scalar)/scalar\*100,*  flags if abs(percent diff) > ***threshold*** |
| Variance Validation  *window*  *center (T/F)*  *num of std dev’s* | * Calculates rolling mean & sd of *(use – rolling mean)* for time period of size ***window*** seconds * Rolling window is either centered if ***center*** is true * ***num of std dev’s*** throws a flag for reads which are this number of rolling sd’s above (spikes) or below (dips) the rolling mean of *(use – rolling mean)* |
| Outlier Validation  *dip spike delta threshold*  *dip spike hours*  *dip spike lookback*  *dip spike max outlier time*  *dip spike min change*  *periodicity num med abs dev*  *periodicity num occur*  *periodicity window* | * Checks for periodicity or dip spike outliers based on if the daily usage is seasonal or not * **Periodicity:** calculates rolling mean w/ default window and centers usage. Groups centered data by day of week & hour. * Calculated median and median average deviation per group * Flags usage if:   *| centered use – group med | >* ***num MAD*** *\* group MAD*  where group med and group MAD based on ***window***   * If num of flags per unique date are > ***num occur***, check this date * **Dip Spike:** calculates rolling mean for ***hours*** window & min periods, if the percent change of rolling mean ***hours*** with starting roll mean as:   *delta roll mean =* *(roll mean(start + hours) / (roll mean(start)) - 1*  for every time in data   * If *delta roll mean >* ***delta threshold***, look backwards from a day ahead to ***lookback*** time before obs (index j) and look for a <= 0 difference, this is the start of spike and roll mean here is right level * If nothing found, use roll mean of time – ***lookback*** as right level***,*** check to see if usage within ***max outlier time*** if <= right level * Otherwise we say it isn’t a spike * If delta roll mean < - ***delta threshold***, repeat process |
| Ramp Up Ramp Down Validation  *compare num days*  *percent threshold*  *usage difference threshold* | * Compare avg daily use between consecutive periods of ***compare num days*** days * *percent variation = (first period ADU – last period ADU)\*100 / first period ADU,* ramp up if > ***percent threshold***, ramp down if less * *usage variation = first period ADU – last period ADU*, same rule applies compared to ***usage difference threshold*** |
| Load Factor IDR Validation  *load factor threshold*  *peak threshold* | * Flags usage if load factor is below ***load factor threshold*** and peak is above ***peak threshold*** * Calculates *load = usage \* 3600 / heartbeat*, use sum, count and max (as peak) * *load factor = sum / peak \* count* |