"Find what matters" ™

SkySpark® Analytic Rules: Combining Full Programmability with a Comprehensive Library of Analytic Functions

Providing the Tools You Need to Address Applications of All Types

SkyFoundry

Version 1.4
Mar 2018

CONTENTS

SkyFoundry

SkySpark® Provides the Tools You Need to Perform Analytics on Virtually Any Type of Data

Lots of people ask us "What analytic functions does SkySpark provide?" Most people ask this because they are familiar with analytics offerings that do a set number of things – in other words they provide a fixed number of standard analytic algorithms. Users may be able to configure these rules, but typically can't modify them or create their own. In many ways this is similar to the early days of automation, where systems offered limited control capabilities and weren't fully programmable.

SkySpark takes a very different approach. SkySpark is a fully programmable analytics platform. This allows building systems experts to implement their own rules that capitalize on their extensive and highly specialized knowledge of building and equipment systems.

But SkySpark offers more than just programming flexibility. The product couples the capability for full user-programmability with an extensive library of standard analytic functions to streamline the process of turning your domain knowledge into SkySpark rules.

The SkySpark® Analytic Function Library

SkySpark includes an extensive library of built-in analytic functions (over 500 at last count and we're always adding more). These functions are tools that perform various types of analytics and data transformations. Many of these can be considered "end-use" rules. For example, if you want to look for conditions where economizer dampers are open during heating or cooling, there is a standard function for that. If you want to find periods of time when heating and cooling are on simultaneously there is a standard function for that as well. These are examples of "application level" analytic functions which are included in SkySpark's "extension libraries". The Energy extension, for example, includes functions to create custom energy baselines, custom energy normalization formulas, conversion of kW history data to kWh, and a host of others. Here are a few examples from the Energy library:

energyBaselinePrevMonth Source

```
energyBaselinePrevMonth(points, dates)
```

Generate a baseline his grid based on last month's data. Data from exactly 28days ago is used to ensure alignment by weekday.

energyNormByArea Source

```
energyNormByArea(hisData)
```

Given a his grid, normalize the values by area in m^2 or ft^2. Each value column must have a `siteRef` which references a site with the `area` tag.

energyNormByDegreeDay Source

```
energyNormByDegreeDay(hisData)
```

Given a his grid, normalize the values by heating or cooling degree-day. Each value column must have a `siteRef` which references a site with the `weatherRef` tag.

The **Energy** extension also includes functions to compare baselines against current conditions to detect deviations and trends, functions to normalize energy data by building size, degree days, and other factors, as well as functions to import and parse **GreenButton™** energy data and functions to integrate with **Energy Star Portfolio Manager™** for automated reporting of energy consumption data and retrieval of energy data already available in Portfolio Manager.

Application-Oriented Libraries

There are many more similar functions in the SkySpark library and most include the actual source code so that you can modify the standard functions to adapt them to your unique project needs. Here's an example of the source code provided for a function in the HVAC library that detects simultaneous heating and cooling:

ahuCoolAndHeat

```
(ahu, dates, minTime: 0min) => do
  // normalize minTime to hours
  minTime = minTime.to(1h)

  // get periods when cooling and heating
  cool: ahuCoolPeriods(ahu, dates)
  heat: ahuHeatPeriods(ahu, dates)

  // compute intersection of those periods
  hisPeriodIntersection([cool, heat])
    .findAll(r => r->v0 >= minTime)
end
```

Other functions provided in the **HVAC** extension include ready-to-go rules such as ahuCoolAndEcon to compute periods when an AHU is in cooling and economizer mode, and ahuCoolHeatCycling to detect short sycling of equipment.

Tools for Data Analytics Needs of All Types

Beyond the application-oriented analytic functions, SkySpark offers a wide range of general-purpose analytic functions designed to look for patterns in data and manipulate and transform data. These core functions are used to assemble analytic rules that meet your specific project needs. Let take a look at some examples.

Want to find toughs or peaks in your dat

hisFindPeaks

```
hisFindPeaks(src)
```

Filter a history grid so that only numeric samples that represent a peak are returned. A peak is defined as a timestamp/value pair where both the previous and next value is smaller. Non-numeric columns are filtered out. If the source grid has multiple numeric value columns, then non-peak cells are set to null. Also see `hisFindTroughs`.

hisFindTroughs

```
hisFindTroughs(src)
```

Filter a history grid so that only numeric samples that represent a trough are returned. A trough is defined as a timestamp/value pair where both the previous and next value is larger. Non-numeric columns are implicitly filtered out. If the source grid has multiple numeric value columns, then non-trough cells are set to null. Also see `hisFindPeaks`.

Want to look for gaps in data records and interpolate across them? SkySpark offers a function for that as well.

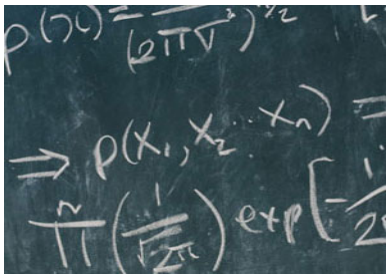
hisInterpolate

```
hisInterpolate(src)
```

Interpolate all the null value cells for each timestamp. See Interpolation for details.

Need to do a linear regression between variables as part of a sophisticated analytic rule? That function is provided as well (see the math function examples below).

Math Functions



SkySpark lets you take advantage of advanced math for sophisticated algorithms. A complete scientific math package is included providing math functions such as linear regression, and trigonometric functions as well as statistical functions including mean bias error, standard deviation and and others:

rootMeanSquareErr

```
rootMeanSquareErr(val, acc, nDegrees: 0)
```

Fold a sample of numbers into their RMSE (root mean square error). The RMSE function determines the RMSE between a sample set and its mean using the n-degrees of freedom RMSE:

```
RMBE = sqrt( Σ(xᵢ - median)² ) / (n - nDegrees)
```

Examples:

```
samples.fold(rootMeanSquareErr)         // unbiased zero degrees of freedom
samples.fold(rootMeanSquareErr(_,_,1))  // 1 degree of freedom
```

standardDeviation

```
standardDeviation(val, acc)
```

Fold a series of numbers into the standard deviation of a *sample*:

```
s = sqrt(Σ (xᵢ - mean)² / (n-1))
```

Example:

```
[4, 2, 5, 8, 6].fold(standardDeviation)
```

fitLinearRegression

```
fitLinearRegression(grid, opts: null)
```

Given a grid of x, y coordinates compute the best fit linear regression equation using the ordinary least squares method. The first column of the grid is used for `x` and the second column is `y`. Any rows without a Number for both x and y are skipped. Any special Numbers (infinity/NaN) are skipped.

Options:

- `x`: column name to use for x if not first column
- `y`: column name to use for y if not second column

The resulting linear equation is:

$$y_i = mx_i + b$$

The equation is returned as a dictionary with these keys:

- `m`: slope of the best fit regression line
- `b`: intercept of the best fit regression line
- `r2`: R^2 coefficient of determination as a number between 1.0 (perfect correlation) and 0.0 (no correlation)
- `xmin`: minimum value of x variable in sample data
- `xmax`: maximum value of x variable in sample data
- `ymin`: minimum value of y variable in sample data
- `ymax`: maximum value of y variable in sample data

Analytic Tools for Common Data Analysis Challenges

One of the most common tools for identifying patterns that represent faults or improper operation is the analytic function that finds intersections in data sets across time – its called hisPeriodIntersection:

hisPeriodIntersection

```
hisPeriodIntersection(grids)
```

Given two or more grids of historical periods, return a new grid with the intersection. The input grids much have a "ts" timestamp column and a second column named anything of period values. The resulting grid has "ts" and "v0" column, where "v0" is duration in hours.

Example:

```
// find periods when both a and b are on
hisPeriodIntersection([a, b])
```

The hisPeriodIntersection function is at the core of many of SkySpark's application-oriented analytic rules such as ahuHeatandCool, which detects simultaneous heating and cooling patterns as well as the ahuHeatAndEcon and ahuCoolAndEcon functions, which detects simultaneous operation of heating or cooling and economizer dampers. **hisPeriodMatrix** looks for even more sophisticated combinations of values and events:

hisPeriodMatrix

```
hisPeriodMatrix(grids)
```

Given two or more grids of historical periods, return a new grid which defines the "matrix" of unique combinations of all the periods merged together. The result is returned as a grid with these columns:

- `ts`: starting timestamp of each unique period
- `dur`: duration in hours of each unique period
- `numTrue`: number of input periods "on" during period
- `v{i}`: true of false indicating if input was "on" during that period

Example, lets assume we have three unit runtime periods a, b, and c:

```
// look for periods where all units are all off
hisPeriodMatrix([a, b, c]).findAll r => r->numTrue == 0

// look for periods where all units are all on
hisPeriodMatrix([a, b, c]).findAll r => r->numTrue == 3

// look for periods where a and b are on, but c is off
hisPeriodMatrix([a, b, c]).findAll r => r->v0 and r->v1 and not r->v2
```

Machine-Learning Functions

The SkySpark library also includes Machine Learning functions that provide support for supervised learning for prediction and forecasting through regression-based approaches, and classification using Support Vector Machine (SVM) techniques.

Overview

The mlExt provides a library of functions to perform machine learning (ML) within SkySpark. There are three broad categories of machine learning algorithms. Currently, the ext provides support for the first category (supervised learning)

1. **Supervised Learning**:

   The algorithm builds a model from input data for which the desired output values *are* known.

2. **Unsupervised Learning**:

   The algorithm builds a model from input data for which the desired output values *are not* known. That is, the algorithm attempts to find the structure in the data.

3. **Reinforcement Learning**:

   The algorithm interacts with a dynamic environment and learns from the consequences of its actions. It is not "trained" with data.

Designed for Flexibility and Efficient Project Implementation

With SkySpark you are not limited to a pre-defined set of analytic functions, AND at the same time you don't have to start from scratch. Similar to the way that most automation systems include standard functions or blocks for PID control loops, schedules, etc., SkySpark provides a rich library of functions needed to perform data analytics.

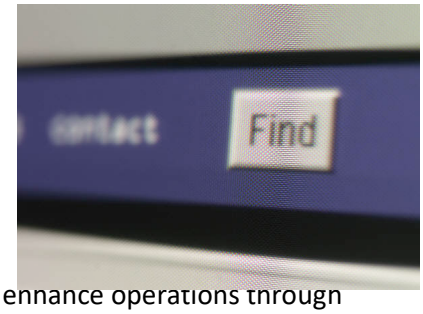
With SkySpark you have the tools you need to convert your domain knowledge into analytic rules that run continuously and automatically against your data to ***find what matters™*** in virtually any application.

With SkySpark® You're In Control

You have the ability to use all of SkySpark's analytic functions to create your own rules and algorithms, combining them with other functions and your own programming logic.

This is a key differentiating feature of SkySpark – you're never limited to a "canned", factory supplied set of analytic functions – you can create rules that capture your experience and systems knowledge to fit the unique needs of your specific applications.

SkySpark® – Analytics for a World of Smart Devices



The past decade has seen dramatic advances in automation systems and smart devices. From IP connected systems using a variety of standard protocols, to support for web services, it is now possible to get the data produced by the wide range of devices found in today's smart devices and equipment systems.

Access to this data opens up new opportunities for the creation of value-added services to help businesses reduce energy consumption and operational costs and to identify opportunities to enhance operations through improved control, and replacement or repair of capital equipment.

Access to the data is just the first step in that journey, however. The new challenge is how to manage and derive value from the exploding amount of data available from these smart and connected devices. SkyFoundry' SkySpark® directly addresses this challenge.

The new frontier is to efficiently manage and analyze data to find what matters.

To learn more and attend a detailed product demonstration contact us at:

info@skyfoundry.com
www.skyfoundry.com

About SkyFoundry

SkyFoundry's mission is to provide software solutions for the age of "the Internet of things". Areas of focus include:

- Building automation and facility management
- Energy management, utility data analytics
- Remote device and equipment monitoring
- Asset management

SkyFoundry products help customers derive value from their investments in smart systems. Learn more at:

www.skyfoundry.com