Parametrized event records data transformations

Anton Antonov
MathematicaForPrediction at WordPress
MathematicaForPrediction at GitHub
MathematicaVsR at GitHub
August-September 2018

Version 0.2

Introduction

In this document we describe transformations of events records data in order to make that data more amenable for the application of Machine Learning (ML) algorithms.

Consider the following **problem formulation** that is done with the next five bullet points.

- From data representing a (most likely very) diverse set of events we want to derive contingency matrices corresponding to each of the variables in that data.
- The events are observations of the values of a certain set of variables for a certain set of entities. Not all entities have events for all variables.
- The observation times do not form a regular time grid.
- Each contingency matrix has rows corresponding to the entities in the data and has columns corresponding to time.
- The software component providing the functionality should allow parametrization and repeated execution. (As in ML classifier training and testing scenarios.)

The phrase “event records data” is used instead of “time series” in order to emphasize that (i) some variables have categorical values, and (ii) the data can be given in some general database form, like transactions long-form.

The required transformations of the event records in the problem formulation above are done through the monad ERTMon, [AAp3]. (The name ERTMon comes from “**E**vent **R**ecords **T**ransformations **Mon**ad”.)

It is assumed that the event records data is put in a form that makes it (relatively) easy to extract time series for the set of entity-variable pairs present in that data.

In brief ERTMon performs the following sequence of transformations.

1. The event records of each entity-variable pair are shifted to adhere to a specified start or end point,
2. The event records for each entity-variable pair are aggregated and normalized with specified functions over a specified regular grid,
3. Entity vs. time interval contingency matrices are made for each combination of variable and aggregation function.

The transformations are specified with a “computation specification” dataset.

Here is an example of an ERTMon pipeline over event records:

```
Out[•]=
ERTMonUnit[] ⟹                                          initialize the monad pipeline
  ERTMonSetEventRecords[eventRecords] ⟹                  ingest event records
  ERTMonSetEntityAttributes[entityAttributes] ⟹          ingest event attributes
  ERTMonSetComputationSpecification[compSpec] ⟹          ingest computation specification
  ERTMonGroupEntityVariableRecords ⟹                     group records per {EntityID, Variable}
  ERTMonRecordGroupsToTimeSeries["MaxTime"] ⟹            align the event records and create time series
  ERTMonAggregateTimeSeries                               aggregate the time series
                                                          (according to the computation specification)
```

The rest of the document describes in detail:

- the structure, format, and interpretation of the event records data and computations specifications,
- the transformations of time series aligning, aggregation, and normalization,
- the software pattern design -- a monad -- that allows sequential specifications of desired transformations.

Concrete examples are given using weather data. See [AAp9].

Package load

The following commands load the packages [AAp1-AAp9]

In[4]:=
```
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/MonadicEventRecordsTransformations.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/MonadicProgramming/MonadicTracing.m"]
Import["https://raw.githubusercontent.com/antononcube/MathematicaForPrediction/master/Misc/WeatherEventRecords.m"]
```

Data load

The data we use is weather data from meteorological stations close to certain major cities. We retrieve the data with the function `WeatherEventRecords` from the package [AAp9].

In[7]:=
```
? WeatherEventRecords
```

WeatherEventRecords[citiesSpec_: {{_String, _String}..}, dateRange:{{_Integer, _Integer, _Integer}, {_Integer, _Integer, _Integer}}, wProps:{_String..} : {"Temperature"}, nStations_Integer : 1] gives an association with event records data.

In[8]:=
```
citiesSpec = {{"Miami", "USA"}, {"Chicago", "USA"}, {"London", "UK"}};
dateRange = {{2017, 7, 1}, {2018, 6, 31}};
wProps = {"Temperature", "MaxTemperature", "Pressure", "Humidity", "WindSpeed"};
res1 = WeatherEventRecords[citiesSpec, dateRange, wProps, 1];
```

In[12]:=
```
citiesSpec = {{"Jacksonville", "USA"}, {"Peoria", "USA"}, {"Melbourne", "Australia"}};
dateRange = {{2016, 12, 1}, {2017, 12, 31}};
res2 = WeatherEventRecords[citiesSpec, dateRange, wProps, 1];
```

Here we assign the obtained datasets to variables we use below:

In[15]:=
```
eventRecords = Join[res1["eventRecords"], res2["eventRecords"]];
entityAttributes = Join[res1["entityAttributes"], res2["entityAttributes"]];
```

Here are the summaries of the datasets `eventRecords` and `entitiyAttributes`:

In[17]:=
```
RecordsSummary[eventRecords]
```

Out[17]=
```
  1 EntityID          2 LocationID             3 ObservationTime           4 Variable                 5 Value
  KNIP      1967      Jacksonville  1967       Min      3 689 539 200      Humidity        2278       Min      -19.33
  WMO95866  1965      Melbourne     1965       1st Qu   3 705 955 200      Temperature     2278       1st Qu   2.78
{ KMDW      1802   ,  Chicago       1802   ,   Mean     3.71474 × 10^9  ,  WindSpeed       2278   ,   Median   16.11     }
  KMFL      1798      Miami         1798       Median   3 715 113 600      MaxTemperature  2252       3rd Qu   26.78
  KGEU      1572      Peoria        1572       3rd Qu   3 723 235 200      Pressure        1474       Mean     153.606
  EGLC      1456      London        1456       Max      3 739 392 000                                 Max      1043.1
```

```
In[18]:= RecordsSummary[entityAttributes]

                                          3 Value
           1 EntityID                     USA           4
           EGLC      2                    Australia     1
           KGEU      2   2 Attribute      Chicago       1
Out[18]= { KMDW      2 , City     6 ,     Jacksonville  1 }
           KMFL      2   Country  6       London        1
           KNIP      2                    Melbourne     1
           WMO95866  2                    (Other)       3
```

Design considerations

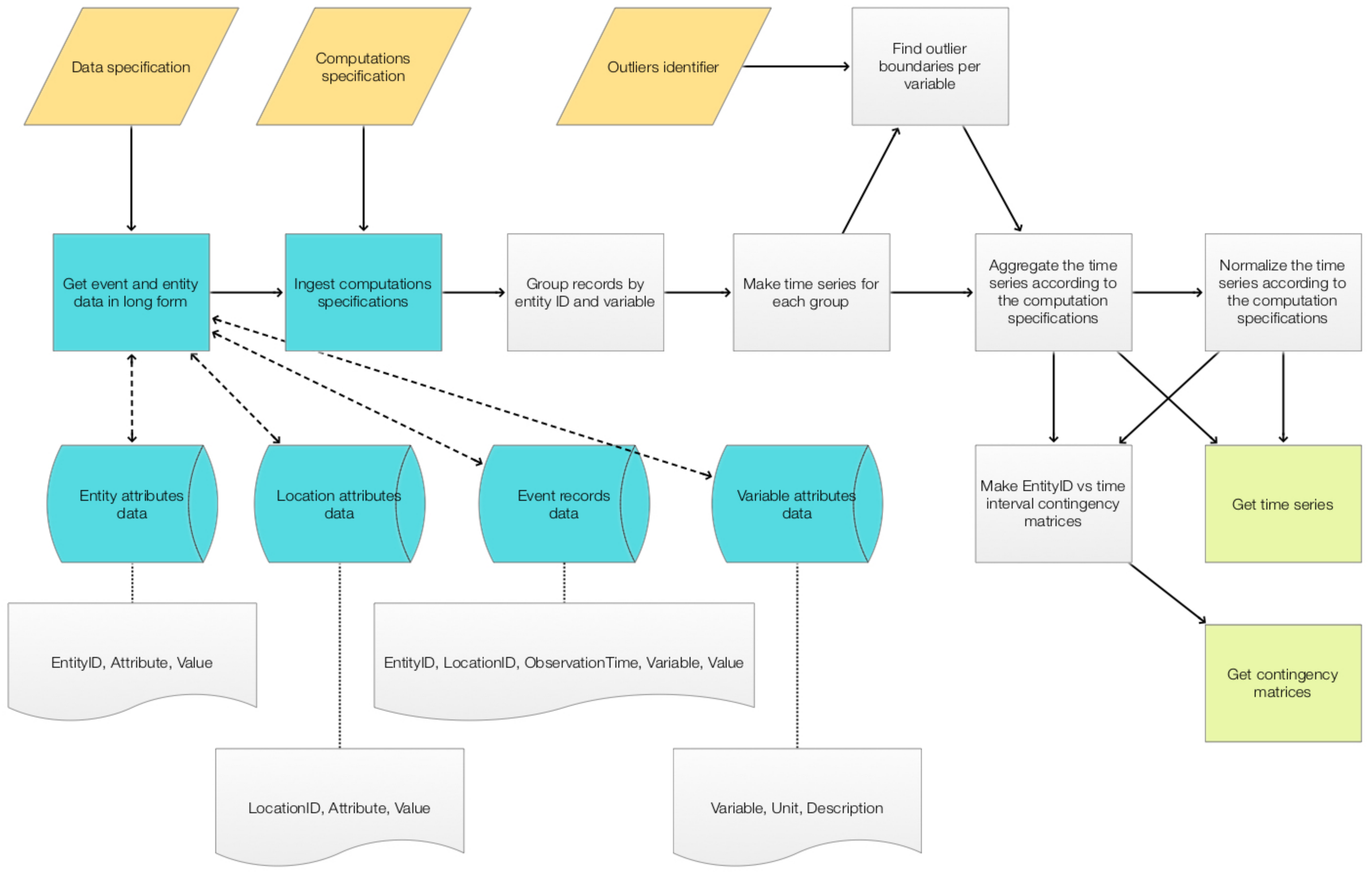
Workflow

The steps of the main event records workflow addressed in this document follow.

1. Ingest event records and entity attributes given in the Star schema style.
2. Ingest a computation specification.
   - **2.1.** Specified are aggregation time intervals, aggregation functions, normalization types and functions.
3. Group event records based on unique entity ID and variable pairs.
   - **3.1.** Additional filtering can be applied using the entity attributes.
4. For each variable find descriptive statistics properties.
   - **4.1.** This is to facilitate normalization procedures.
   - **4.2.** Optionally, for each variable find outlier boundaries.
5. Align each group of records to start or finish at some specified point.
   - **5.1.** For each variable we want to impose a regular time grid.
6. From each group of records produce a time series.
7. For each time series do prescribed aggregation and normalization.
   - **7.1.** The variable that corresponds to each group of records has at least one (possibly several) computation specifications.
8. Make a contingency matrix for each time series obtained in the previous step.
   - **8.1.** The contingency matrices have entity ID’s as rows, and time intervals enumerating values of time intervals.

The following flow-chart corresponds to the list of steps above.

Out[19]=


Data specification
Computations specification
Outliers identifier
Find outlier boundaries per variable
Get event and entity data in long form
Ingest computations specifications
Group records by entity ID and variable
Make time series for each group
Aggregate the time series according to the computation specifications
Normalize the time series according to the computation specifications
Entity attributes data
Location attributes data
Event records data
Variable attributes data
Make EntityID vs time interval contingency matrices
Get time series
EntityID, Attribute, Value
EntityID, LocationID, ObservationTime, Variable, Value
Get contingency matrices
LocationID, Attribute, Value
Variable, Unit, Description

A corresponding monadic pipeline is given in the section “Larger example pipeline”.

Feature engineering perspective

The workflow above describes a way to do feature engineering over a collection of event records data. For a given entity ID and a variable we derive several different time series.

Couple of examples follow.

- One possible derived feature (times series) is for each entity-variable pair we make time series of the hourly mean value in each of the eight most recent hours for that entity. The mean values are normalized by the average values of the records corresponding to that entity-variable pair.
- Another possible derived feature (time series) is for each entity-variable pair to make a time series with the number of outliers in the each half-hour interval, considering the most recent 20 half-hour intervals. The outliers are found by using outlier boundaries derived by analyzing all values of the corresponding variable, across all entities.

From the examples above -- and some others -- we conclude that for each feature we want to be able to specify:

- aggregation interval length,
- aggregation function (to be applied in each interval),
- maximum history length (say from the most recent observation),
- normalization function (per entity, per cohort of entities, per variable),
- conversion of categorical values into numerical ones.

Repeated execution

We want to be able to do repeated executions of the specified workflow steps.

Consider the following scenario. After the event records data is converted to a entity-vs-feature contingency matrix, we use that matrix to train and test a classifier. We want to find the combination of features that gives the best classifier results. For that reason we want to be able to easily and systematically change the computation specifications (interval size, aggregation and normalization functions, etc.) With different computation specifications we obtain different entity-vs-feature contingency matrices, that would have different performance with different classifiers.

Using the classifier training and testing scenario we see that there is another repeated execution perspective: after the feature engineering is done over the training data, we want to be able to execute exactly the same steps over the test data. Note that with the training data we find certain global or cohort normalization values and outlier boundaries that have to be used over the test data. (Not derived from the test data.)

Event records data design

The data is structured to follow the style of Star schema. We have event records dataset (table) and entity attributes dataset (table).

The structure datasets (tables) proposed satisfy a wide range of modeling data requirements. (Medical and financial modeling included.)

Entity event data

The entity event data has the columns “EntityID”, “LocationID”, “ObservationTime”, “Variable”, “Value”.

```
In[21]:= RandomSample[eventRecords, 6]
```

Out[21]=

EntityID	LocationID	ObservationTime	Variable	Value
WMO95866	Melbourne	3693513600	Humidity	0.356
KNIP	Jacksonville	3691958400	Pressure	1019.2
WMO95866	Melbourne	3704054400	Temperature	13.72
KNIP	Jacksonville	3719433600	Pressure	1023.5
KNIP	Jacksonville	3697574400	Temperature	12.33
KNIP	Jacksonville	3723580800	Temperature	7.94

Most events can be described through “Entity event data”. The entities can be anything that produces a set of event data: financial transactions, vital sign monitors, wind speed sensors, chemical concentrations sensors.

The locations can be anything that gives the events certain "spatial" attributes: medical units in hospitals, sensors geo-locations, tiers of financial transactions.

Entity attributes data

The entity attributes dataset (table) has attributes (immutable properties) of the entities. (Like, gender and race for people, longitude and latitude for wind speed sensors.)

```
In[22]:= entityAttributes⟦1 ;; 6⟧
```

Out[22]=

EntityID	Attribute	Value
KMFL	City	Miami
KMFL	Country	USA
KMDW	City	Chicago
KMDW	Country	USA
EGLC	City	London
EGLC	Country	UK

Example

For example, here we take all weather stations in USA:

```
In[23]:= ws = Normal[entityAttributes[Select[#Attribute == "Country" && #Value == "USA" &], "EntityID"]]
Out[23]= {KMFL, KMDW, KNIP, KGEU}
```

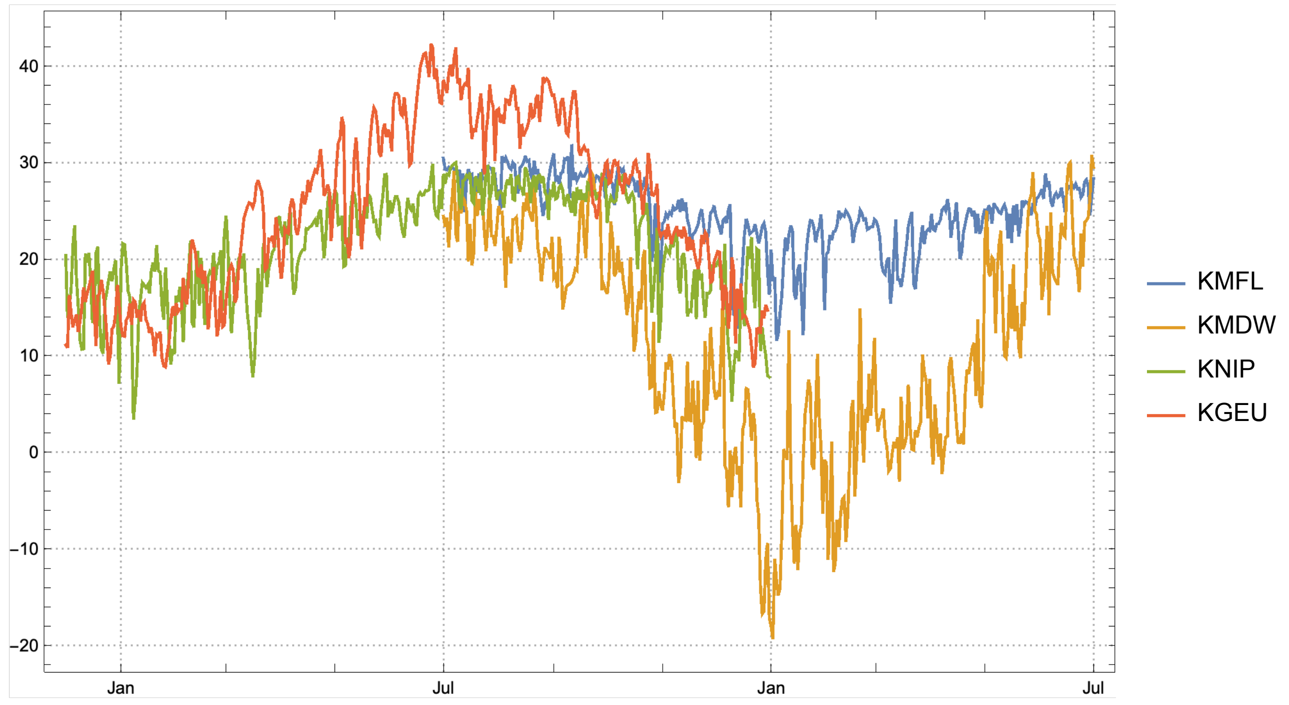
Here we take all temperature event records for those weather stations:

```
In[24]:= srecs = eventRecords[Select[#Variable == "Temperature" && MemberQ[ws, #EntityID] &]];
```

And here plot the corresponding time series obtained by grouping the records by station (entity ID’s) and taking the columns “ObservationTime” and ”Value”:

```
In[25]:= grecs = Normal@GroupBy[srecs, #EntityID &][All, All, {"ObservationTime", "Value"}];
         DateListPlot[grecs, ImageSize → Large, PlotTheme → "Detailed"]
```

Out[26]=



Monad elements

This section goes through the steps of the general ERTMon workflow.

Monad unit

The monad is initialized with ERTMonUnit.

```
In[27]:= ERTMonUnit[]

Out[27]= ERTMon[None, <| |>]
```

Ingesting event records and entity attributes

The event records dataset (table) and entity attributes dataset (table) are set with corresponding setter functions. Alternatively, they can be read from files in a specified directory.

```
In[28]:= p =
           ERTMonUnit[] ⟹
            ERTMonSetEventRecords[eventRecords] ⟹
            ERTMonSetEntityAttributes[entityAttributes] ⟹
            ERTMonEchoDataSummary;
```

```
                                                                                                                                                                  3 Value
                                 1 EntityID        2 LocationID        3 ObservationTime          4 Variable             5 Value                       1 EntityID                      USA           4
                                 KNIP      1967    Jacksonville 1967   Min     3 689 539 200      Humidity       2278    Min     -19.33                EGLC      2   2 Attribute     Australia     1
» Data summary: ⟨|eventRecords → {WMO95866 1965,   Melbourne    1965,  1st Qu  3 705 955 200,     Temperature    2278,   1st Qu  2.78     }, entityAttributes → {KGEU 2, City 6,        Chicago       1 }|⟩
                                 KMDW      1802    Chicago      1802   Mean    3.71474 × 10^9     WindSpeed      2278    Median  16.11                 KMDW      2   Country 6       Jacksonville  1
                                 KMFL      1798    Miami        1798   Median  3 715 113 600      MaxTemperature 2252    3rd Qu  26.78                 KMFL      2                   London        1
                                 KGEU      1572    Peoria       1572   3rd Qu  3 723 235 200      Pressure       1474    Mean    153.606               KNIP      2                   Melbourne     1
                                 EGLC      1456    London       1456   Max     3 739 392 000                             Max     1043.1                WMO95866  2                   (Other)       3
```

Computation specification

Using the package [AAp3] we can create computation specification dataset. Below is given an example of constructing a fairly complicated

```
In[29]:= EmptyComputationSpecificationRow[]
Out[29]= ⟨|Variable → Missing[], Explanation → , MaxHistoryLength → 3600, AggregationTimeInterval → 60, AggregationFunction → Mean, NormalizationScope → Entity, NormalizationFunction → None|⟩
```

```
In[33]:= compSpecRows = Join[EmptyComputationSpecificationRow[], <|"Variable" → #, "AggregationTimeInterval" → 3 * 24 * 3600, "MaxHistoryLength" → 12 * 24 * 3600,
               "AggregationFunction" → "Mean", "NormalizationScope" → "Entity", "NormalizationFunction" → Mean|>] & /@ Union[Normal[eventRecords[All, "Variable"]]];
         compSpecRows =
           Join[
            compSpecRows, Join[EmptyComputationSpecificationRow[], <|"Variable" → #, "AggregationTimeInterval" → 3 * 24 * 3600, "MaxHistoryLength" → 12 * 24 * 3600,
                "AggregationFunction" → "Range", "NormalizationScope" → "Country", "NormalizationFunction" → Mean|>] & /@ Union[Normal[eventRecords[All, "Variable"]]],
            Join[EmptyComputationSpecificationRow[], <|"Variable" → #, "AggregationTimeInterval" → 3 * 24 * 3600, "MaxHistoryLength" → 12 * 24 * 3600,
                "AggregationFunction" → "OutliersCount", "NormalizationScope" → "Variable"|>] & /@ Union[Normal[eventRecords[All, "Variable"]]]
           ];
```

```
In[35]:= wCompSpec = ProcessComputationSpecification[Dataset[compSpecRows]][SortBy[#Variable &]]
```

Out[35]=

	Variable	Explanation	MaxHistoryLength	AggregationTimeInterval	AggregationFunction	NormalizationScope	NormalizationFunction
Humidity.Mean	Humidity		1 036 800	259 200	Mean	Entity	Mean
Humidity.OutliersCount	Humidity		1 036 800	259 200	OutliersCount	Variable	None
Humidity.Range	Humidity		1 036 800	259 200	Range	Country	Mean
MaxTemperature.Mean	MaxTemperature		1 036 800	259 200	Mean	Entity	Mean
MaxTemperature.OutliersCount	MaxTemperature		1 036 800	259 200	OutliersCount	Variable	None
MaxTemperature.Range	MaxTemperature		1 036 800	259 200	Range	Country	Mean
Pressure.Mean	Pressure		1 036 800	259 200	Mean	Entity	Mean
Pressure.OutliersCount	Pressure		1 036 800	259 200	OutliersCount	Variable	None
Pressure.Range	Pressure		1 036 800	259 200	Range	Country	Mean
Temperature.Mean	Temperature		1 036 800	259 200	Mean	Entity	Mean
Temperature.OutliersCount	Temperature		1 036 800	259 200	OutliersCount	Variable	None
Temperature.Range	Temperature		1 036 800	259 200	Range	Country	Mean
WindSpeed.Mean	WindSpeed		1 036 800	259 200	Mean	Entity	Mean
WindSpeed.OutliersCount	WindSpeed		1 036 800	259 200	OutliersCount	Variable	None
WindSpeed.Range	WindSpeed		1 036 800	259 200	Range	Country	Mean

Alternatively, computation specification can be created and filled-in as a CSV file and read into the monad.

Descriptive statistics (per variable)

Grouping event records by entity-variable pairs

Time series creation, alignment, and aggregation

Normalization

With “normalization” we mean that the values of a given time series values are divided (normalized) with a descriptive statistic derived from a specified set of values. The specified set of values is given with the parameter “NormalizationScope” in computation specification.

At the normalization stage each time series is associated with an entity ID and variable.

Normalization is done at three different scopes: “entity”, “attribute”, and “variable”.

Given a time series *T*(*i*, *var*) corresponding to entity ID *i* and a variable *var* we define the normalization values for the different scopes in the following ways.

- Normalization with scope “entity” means that the descriptive statistic is derived from the values of *T*(*i*, *var*) only.
- Normalization with scope attribute means that
  - from the entity attributes dataset we find attribute value that corresponds to *i*,
  - next we find all entity ID's that are associated with the same attribute value,
  - next we find value of normalization descriptive statistic using the time series that correspond to the variable *var* and the entity ID’s found in the previous step.
- Normalization with scope ”variable” means that the descriptive statistic is derived from the values of all time series corresponding to *var*.

Note that the scope “entity” is the most granular, and the scope “variable” is the coarsest.

Making contingency matrices

The contingency matrices are given as SSparseMatrix objects, [AAp7].

Larger example pipeline

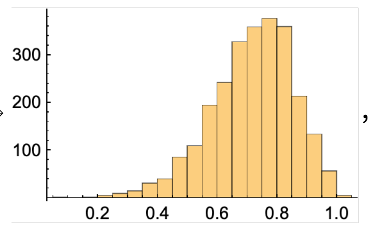
The pipeline shown in this section utilizes all workflow functions of ERTMon. The used weather data and computation specification are described above.

```
ERTMonUnit[] ⟹                                                                          initialize the monad pipeline
  ERTMonSetEventRecords[eventRecords] ⟹                                                  ingest event records
  ERTMonSetEntityAttributes[entityAttributes] ⟹                                          ingest event attributes
  ERTMonEchoDataSummary ⟹                                                                show data summary
  ERTMonSetComputationSpecification[compSpec] ⟹                                          ingest computation specification
  ERTMonGroupEntityVariableRecords ⟹                                                     group records per {EntityID, Variable}
  ERTMonFindVariableDistributions[Histogram[#1, ImageSize → Small] &] ⟹                  find variable distributions
  ERTMonEchoFunctionValue["Variable distributions:", #1 &] ⟹                             echo found distributions
  ERTMonFindVariableOutlierBoundaries[SPLUSQuartileIdentifierParameters] ⟹               find outliers boundaries for each variable
  ERTMonEchoFunctionValue["Outlier boundaries:", #1 &] ⟹                                 echo found outliers boundaries
  ERTMonRecordGroupsToTimeSeries["MaxTime"] ⟹                                            align the event records and create time series
  ERTMonAggregateTimeSeries ⟹                                                            aggregate the time series
                                                                                          (according to the computation specification)
  ERTMonMakeContingencyMatrices ⟹                                                        make contingency matrices
  ERTMonEchoFunctionValue["Contingency matrices:", (MatrixPlot[ToSSparseMatrix[#1], ImageSize → Small] &) /@ #1 &]   plot contingency matrices
```

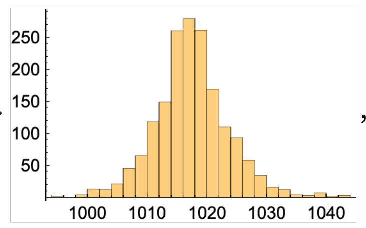
Out[•]=

```
                                   1 EntityID                      3 ObservationTime             4 Variable               5 Value                                    1 EntityID               3 Value
                                   WMO95866  1456  2 LocationID    Min      3 707 856 000                                     Min     -19.33                             EGKB     2               USA        4
                                   KMFL      1445  Miami      2890 1st Qu   3 715 632 000        Humidity      2554           1st Qu  0.853                              EGLC     2  2 Attribute   Australia  2
» Data summary: ⟨|eventRecords → { KMIA      1445  Chicago    2877 , Median 3 723 580 800 ,      Temperature   2554 ,         Median  13.39  }, entityAttributes → {    KMDW     2  City     8 , Chicago    2 }|⟩
                                   KMDW      1440  London     2178 Mean     3.72358 × 10^9       WindSpeed     2554           3rd Qu  25.11                              KMFL     2  Country  8   London     2
                                   KORD      1437  Melbourne  1456 3rd Qu   3 731 443 200        Pressure      1739           Mean    196.667                            KMIA     2               Melbourne  2
                                   EGLC      1092                  Max      3 739 392 000                                     Max     1043.1                             KORD     2               Miami      2
                                   EGKB      1086                                                                                                                        (Other)  4               UK         2
```

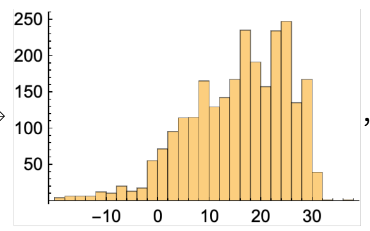
» Variable distributions: ⟨|Humidity →



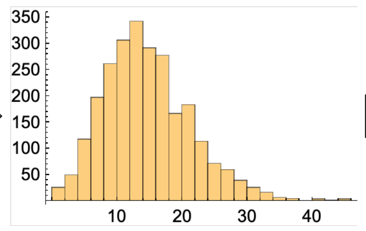
, Pressure →



, Temperature →



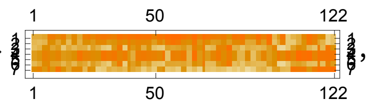
, WindSpeed →



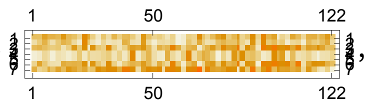
|⟩

```
» Outlier boundaries: ⟨|Humidity → {0.359, 1.087}, Pressure → {1003.8, 1031.}, Temperature → {-12.765, 45.435}, WindSpeed → {-3.23, 31.57}|⟩
```

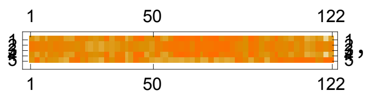
» Contingency matrices: ⟨|Humidity.Mean →



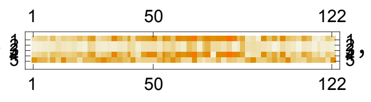
, Humidity.Range →



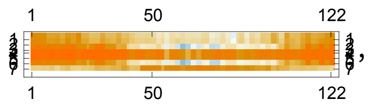
, Pressure.Mean →



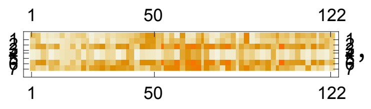
, Pressure.Range →



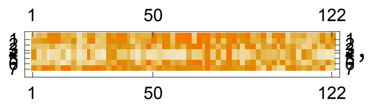
, Temperature.Mean →



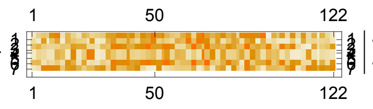
, Temperature.Range →



, WindSpeed.Mean →



, WindSpeed.Range →



|⟩

References

Packages

[AAp1] Anton Antonov, State monad code generator Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/StateMonadCodeGenerator.m .

[AAp2] Anton Antonov, Monadic tracing Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicTracing.m .

[AAp3] Anton Antonov, Monadic Event Records Transformations Mathematica package, (2018), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicEventRecordsTransformations.m .

[AAp4] Anton Antonov, MathematicaForPrediction utilities, (2014), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/MathematicaForPredictionUtilities.m .

[AAp5] Anton Antonov, Cross tabulation implementation in Mathematica, (2017), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/CrossTabulate.m .

[AAp6] Anton Antonov, Implementation of one dimensional outlier identifying algorithms in Mathematica, (2013), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/OutlierIdentifiers.m .

[AAp7] Anton Antonov, SSparseMatrix Mathematica package, (2018), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/SSparseMatrix.m .

[AAp8] Anton Antonov, Monadic contextual classification Mathematica package, (2017), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/MonadicProgramming/MonadicContextualClassification.m .

[AAp9] Anton Antonov, Weather event records data Mathematica package, (2018), MathematicaForPrediction at GitHub.
URL: https://github.com/antononcube/MathematicaForPrediction/blob/master/Misc/WeatherEventRecords.m .

Documents

[AA1] Anton Antonov, A monad for classification workflows, (2018), MathematicaForPrediction at WordPress.
URL: https://mathematicaforprediction.wordpress.com/2018/05/15/a-monad-for-classification-workflows .