

Parametrized event records data transformations

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

Version 1.0

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** (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 **M**onad”.)

The monad code generation and utilization is explained in [AA1] and implemented with [AAp1].

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[*n*]=

ERTMonUnit[] =>
ERTMonSetEventRecords[eventRecords] =>
ERTMonSetEntityAttributes[entityAttributes] =>
ERTMonSetComputationSpecification[compSpec] =>
ERTMonGroupEntityVariableRecords =>
ERTMonEntityVariableGroupsToTimeSeries["MaxTime"] =>
ERTMonAggregateTimeSeries

initialize the monad pipeline
ingest event records
ingest event attributes
ingest computation specification
group records per {EntityID, Variable}
align the event records and create time series
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[54]:= 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[ ]:= ? 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[ ]:= 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[ ]:= 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[ ]:= eventRecords = Join[res1["eventRecords"], res2["eventRecords"]];
entityAttributes = Join[res1["entityAttributes"], res2["entityAttributes"]];
```

Here are the summaries of the datasets eventRecords and entityAttributes:

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

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

$In[*]:=$ RecordsSummary[entityAttributes]

1 EntityID		2 Attribute		3 Value	
EGLC	2			USA	4
KGEU	2			Australia	1
KMDW	2	City	6	Chicago	1
				Jacksonville	1
KMFL	2	Country	6	London	1
KNIP	2			Melbourne	1
WM095866	2			(Other)	3

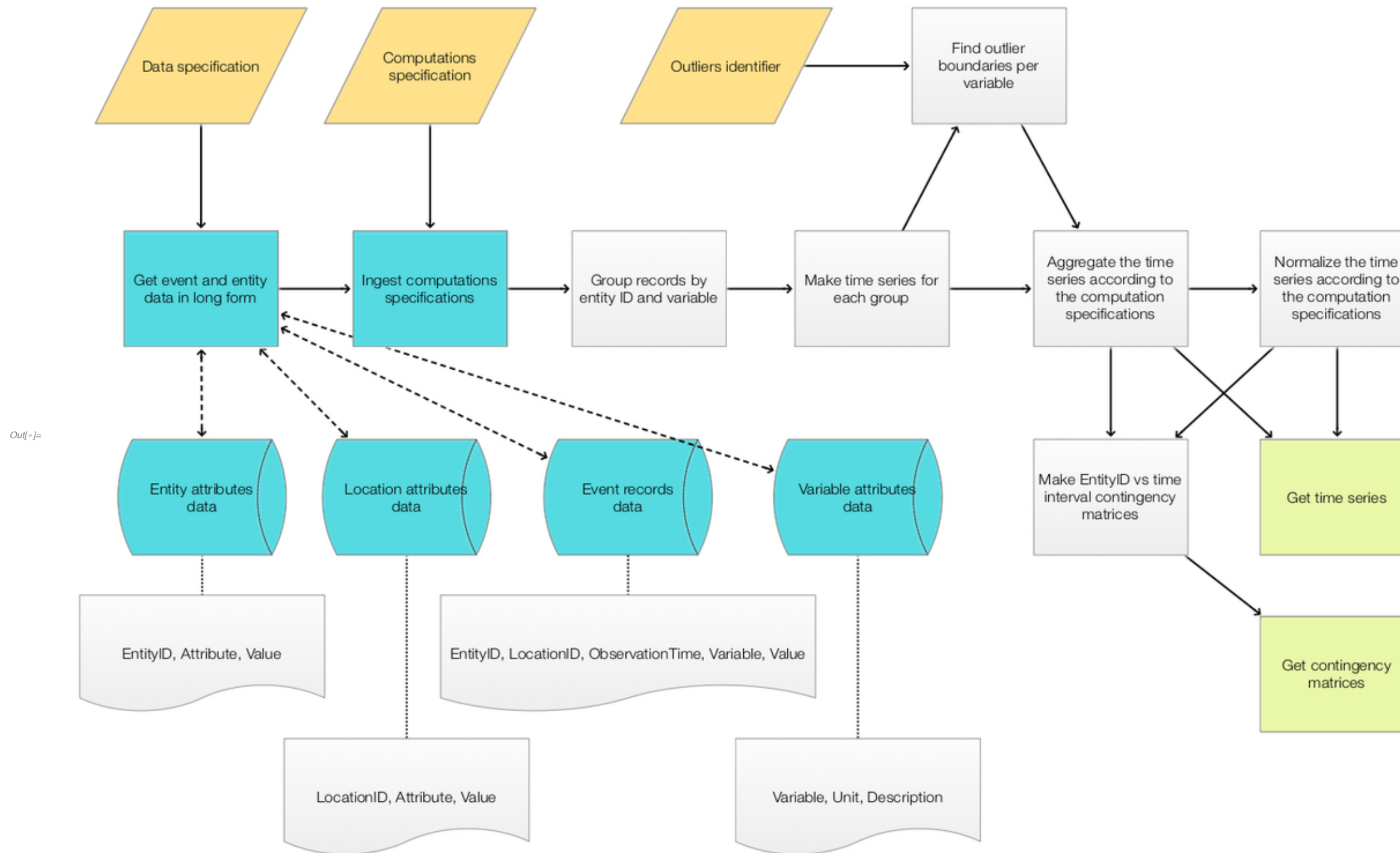
Design considerations

Workflow

The steps of the main event records transformations 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.



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:

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

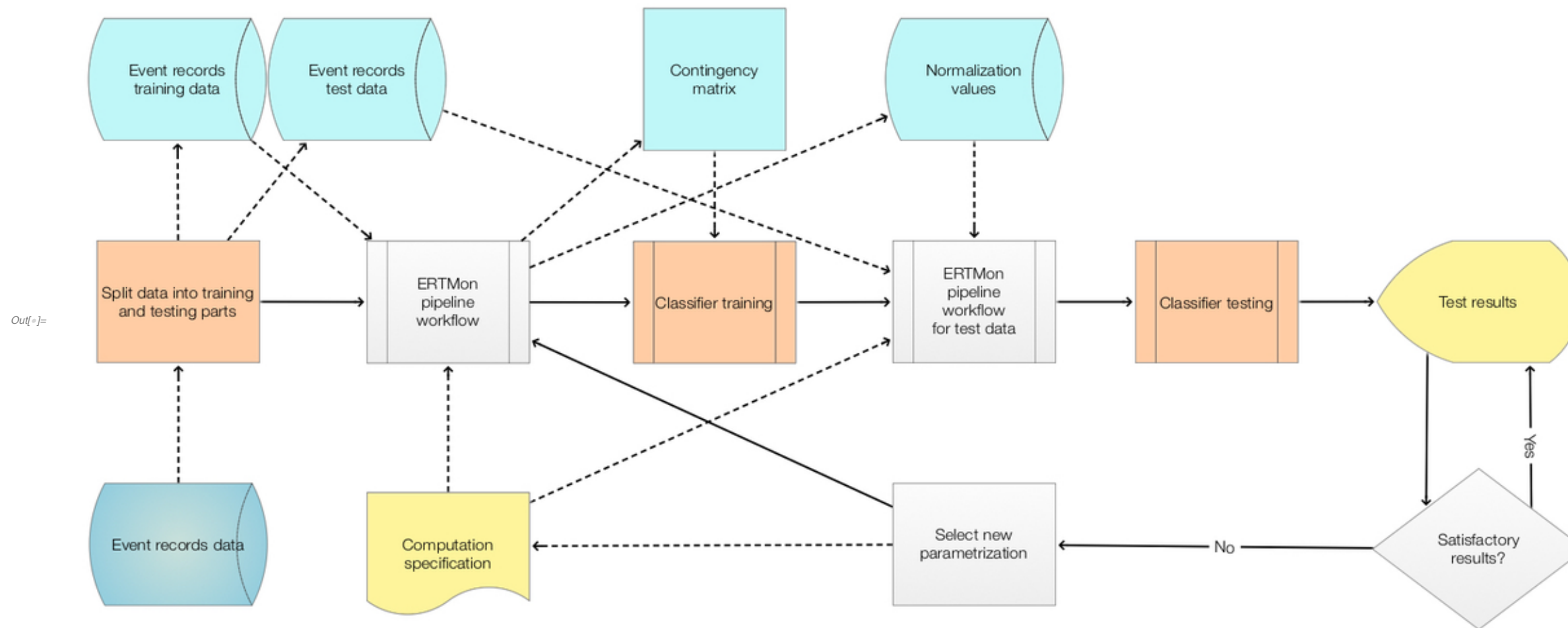
Repeated execution

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.)

The following diagram further describes the repeated execution workflow.



Further discussion of making and using ML classification workflows through the monad software design pattern can be found in [AA2].

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[*]:= RandomSample[eventRecords, 6]

Out[*]=

EntityID	LocationID	ObservationTime	Variable	Value
WMO95866	Melbourne	3 713 731 200	WindSpeed	20.74
EGLC	London	3 724 272 000	Humidity	0.698
KMDW	Chicago	3 721 507 200	Pressure	1016.8
WMO95866	Melbourne	3 704 918 400	Pressure	1012
KNIP	Jacksonville	3 691 526 400	Temperature	16.5
KNIP	Jacksonville	3 695 068 800	WindSpeed	8.15

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[*]:= entityAttributes[[1 ;; 6]]

Out[*]=

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[*]:= ws = Normal[entityAttributes[Select[#Attribute == "Country" && #Value == "USA" &], "EntityID"]]

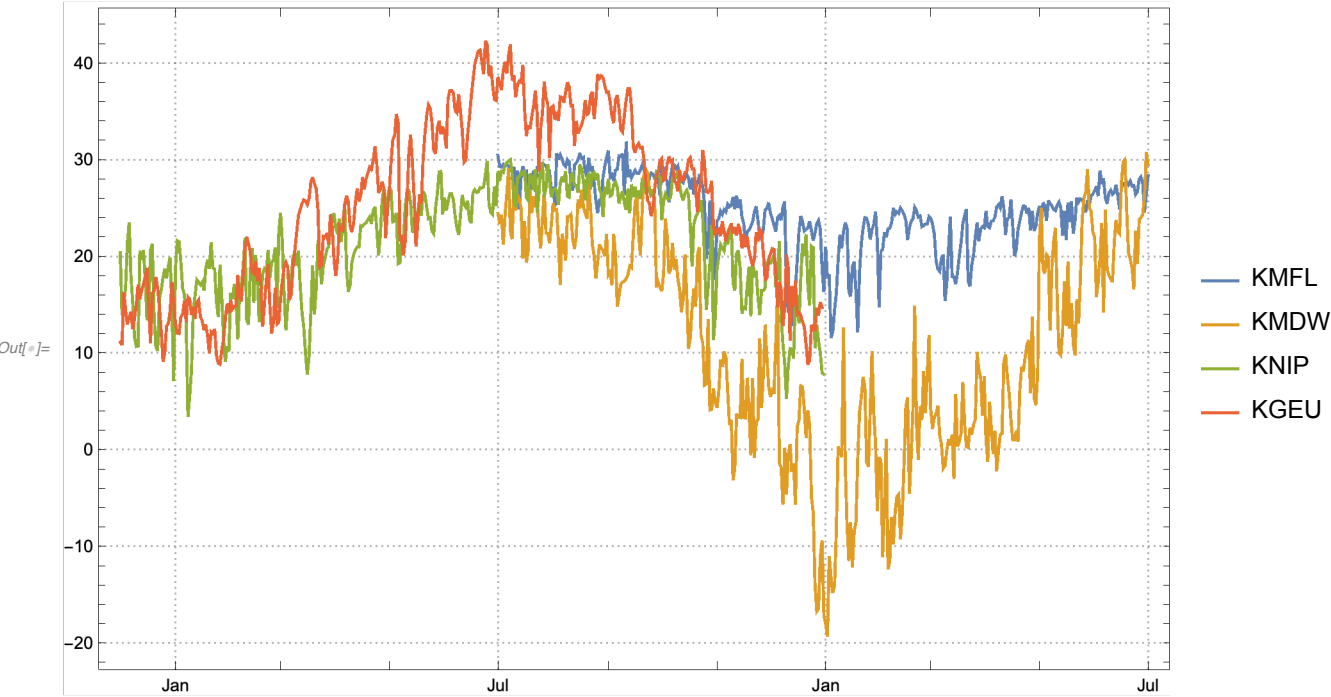
Out[*]= { KMFL, KMDW, KNIP, KGEU }

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

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



Monad elements

This section goes through the steps of the general ERTMon workflow. For didactic purposes each sub-section changes the pipeline assigned to the variable `p`. Of course all functions can be chained into one big pipeline as shown in the section “Larger example pipeline”.

Monad unit

The monad is initialized with `ERTMonUnit`.

```
In[102]:= ERTMonUnit[]
Out[102]:= 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[103]:= p =
ERTMonUnit[] =>
ERTMonSetEventRecords[eventRecords] =>
ERTMonSetEntityAttributes[entityAttributes] =>
ERTMonEchoDataSummary;
```

» Data summary: $\langle \left| \text{eventRecords} \rightarrow \left\{ \begin{array}{llll} \text{1 EntityID} & & \text{2 LocationID} & \text{3 ObservationTime} \\ \text{KMFL21} & 1590 & \text{Tampa} & 23\,695 \\ \text{KTPA21} & 1588 & \text{Miami} & 23\,685 \\ \text{KMDW21} & 1581 & \text{, Jacksonville} & 23\,678 \\ \text{KNIP21} & 1580 & \text{Chicago} & 23\,587 \\ \text{KFTY21} & 1575 & \text{Atlanta} & 23\,352 \\ \text{KNIP17} & 1513 & \text{London} & 17\,628 \\ \text{(Other)} & 126\,198 & & \end{array} \right. , \begin{array}{ll} \text{4 Variable} & \text{5 Value} \\ \text{Temperature} & 35\,608 \\ \text{WindSpeed} & 35\,608 \\ \text{Humidity} & 35\,587 \\ \text{Pressure} & 28\,822 \end{array} , \begin{array}{ll} \text{Min} & -19.33 \\ \text{1st Qu} & 0.867 \\ \text{Median} & 13.33 \\ \text{3rd Qu} & 27.39 \\ \text{Mean} & 224.437 \\ \text{Max} & 1043.1 \end{array} \right\} , \text{entityAttributes} \rightarrow \left\{ \begin{array}{llll} \text{1 EntityID} & & \text{2 Attribute} & \text{3 Value} \\ \text{EGLC1} & 3 & & \text{USA} \\ \text{EGLC10} & 3 & & \text{Atlanta} \\ \text{EGLC11} & 3 & \text{City} & 144 \\ \text{EGLC12} & 3 & \text{Country} & 144 \\ \text{EGLC13} & 3 & \text{StationID} & 144 \\ \text{EGLC14} & 3 & & \text{Jacksonville} \\ \text{(Other)} & 414 & & \text{KFTY} \\ & & & \text{(Other)} \end{array} \right\} \rangle$

Computation specification

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

The package EmptyComputationSpecificationRow can be used to construct the rows of the specification.

```
In[104]:= EmptyComputationSpecificationRow[]
Out[104]= <|Variable -> Missing[], Explanation -> , MaxHistoryLength -> 3600, AggregationIntervalLength -> 60, AggregationFunction -> Mean, NormalizationScope -> Entity, NormalizationFunction -> None|>

In[105]:= compSpecRows = Join[EmptyComputationSpecificationRow[], <|"Variable" -> #, "MaxHistoryLength" -> 60 * 24 * 3600, "AggregationIntervalLength" -> 2 * 24 * 3600,
"AggregationFunction" -> "Mean", "NormalizationScope" -> "Entity", "NormalizationFunction" -> "Mean"|>] & /@ Union[Normal[eventRecords[All, "Variable"]]];
compSpecRows =
Join[
compSpecRows, Join[EmptyComputationSpecificationRow[], <|"Variable" -> #, "MaxHistoryLength" -> 60 * 24 * 3600, "AggregationIntervalLength" -> 2 * 24 * 3600,
"AggregationFunction" -> "Range", "NormalizationScope" -> "Country", "NormalizationFunction" -> "Mean"|>] & /@ Union[Normal[eventRecords[All, "Variable"]]],
Join[EmptyComputationSpecificationRow[], <|"Variable" -> #, "MaxHistoryLength" -> 60 * 24 * 3600, "AggregationIntervalLength" -> 2 * 24 * 3600,
"AggregationFunction" -> "OutliersCount", "NormalizationScope" -> "Variable"|>] & /@ Union[Normal[eventRecords[All, "Variable"]]]
];
```

The constructed rows are assembled into a dataset (with Dataset). The function ProcessComputationSpecification is used to convert a user-made specification dataset into a form used by ERTMon.

In[107]:= **wCompSpec = ProcessComputationSpecification[Dataset[compSpecRows]][SortBy[#Variable &]]**

Out[107]=

	Variable	Explanation	MaxHistoryLength	AggregationIntervalLength	AggregationFunction	NormalizationScope	NormalizationFunction
Humidity.Mean	Humidity		5 184 000	172 800	Mean	Entity	Mean
Humidity.OutliersCount	Humidity		5 184 000	172 800	OutliersCount	Variable	None
Humidity.Range	Humidity		5 184 000	172 800	Range	Country	Mean
Pressure.Mean	Pressure		5 184 000	172 800	Mean	Entity	Mean
Pressure.OutliersCount	Pressure		5 184 000	172 800	OutliersCount	Variable	None
Pressure.Range	Pressure		5 184 000	172 800	Range	Country	Mean
Temperature.Mean	Temperature		5 184 000	172 800	Mean	Entity	Mean
Temperature.OutliersCount	Temperature		5 184 000	172 800	OutliersCount	Variable	None
Temperature.Range	Temperature		5 184 000	172 800	Range	Country	Mean
WindSpeed.Mean	WindSpeed		5 184 000	172 800	Mean	Entity	Mean
WindSpeed.OutliersCount	WindSpeed		5 184 000	172 800	OutliersCount	Variable	None
WindSpeed.Range	WindSpeed		5 184 000	172 800	Range	Country	Mean

The computation specification is set to the monad with the function `ERTMonSetComputationSpecification`.

Alternatively, a computation specification can be created and filled-in as a CSV file and read into the monad. (Not described here.)

Grouping event records by entity-variable pairs

With the function `ERTMonGroupEntityVariableRecords` we group the event records by the found unique entity-variable pairs. Note that in the pipeline below we set the computation specification first.

In[108]:= **p =**
p ==>
ERTMonSetComputationSpecification[wCompSpec] ==>
ERTMonGroupEntityVariableRecords;

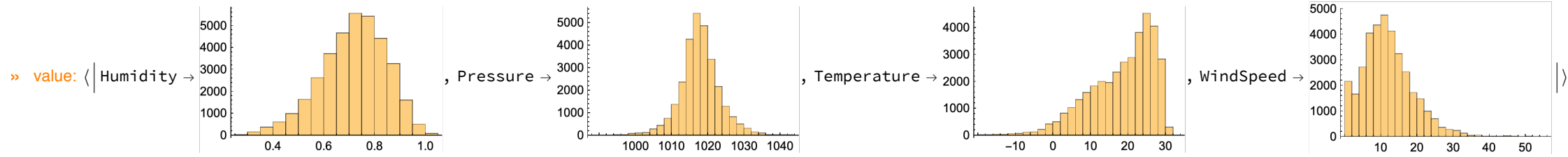
Descriptive statistics (per variable)

After the data is ingested into the monad and the event records are grouped per entity-variable pairs we can find certain descriptive statistics for the data. This is done with the general function `ERTMonComputeVariableStatistic` and the specialized function `ERTMonFindVariableOutlierBoundaries`.

In[109]:= **p ==> ERTMonComputeVariableStatistic[RecordsSummary] ==> ERTMonEchoValue;**

» value: { { 1 column 1
Min 0.255
1st Qu 0.635
Mean 0.716602
Median 0.729
3rd Qu 0.809
Max 1. } Humidity -> { Mean 0.716602 }, Pressure -> { Median 1017.6
Mean 1017.8
3rd Qu 1020.7
Max 1043.1 } Temperature -> { Mean 18.3807
Median 20.39
3rd Qu 25.22
Max 32.28 } WindSpeed -> { Median 11.11
Mean 11.913
3rd Qu 15.74
Max 54.82 } }

```
In[110]:= p ⇒ ERTMonComputeVariableStatistic ⇒ ERTMonEchoValue;
```



```
In[111]:= p ⇒ ERTMonComputeVariableStatistic[TakeLargest[#, 3] &] ⇒ ERTMonEchoValue;
```

» value: <| Humidity → {1., 1., 1.}, Pressure → {1043.1, 1043.1, 1043.1}, Temperature → {32.28, 32.28, 32.28}, WindSpeed → {54.82, 54.82, 54.82} |>

Finding the variables outlier boundaries

The finding of outliers counts and fractions can be specified in the computation specification. Because of this there is a specialized function for outlier finding `ERTMonFindVariableOutlierBoundaries`. That function makes the association of the found variable outlier boundaries (i) to be the pipeline value and (ii) to be the value of context key “variableOutlierBoundaries”. The outlier boundaries are found using the functions of the package [AAp6].

If no argument is specified `ERTMonFindVariableOutlierBoundaries` uses the Hampel identifier (`HampelIdentifierParameters`).

```
In[112]:= p ⇒ ERTMonFindVariableOutlierBoundaries ⇒ ERTMonEchoValue;
```

» value: <| Humidity → {0.600014, 0.857986}, Pressure → {1013.15, 1022.05}, Temperature → {11.9095, 28.8705}, WindSpeed → {4.79412, 17.4259} |>

```
In[113]:= Keys[p ⇒ ERTMonFindVariableOutlierBoundaries ⇒ ERTMonTakeContext]
```

```
Out[113]:= {eventRecords, entityAttributes, computationSpecification, entityVariableRecordGroups, variableOutlierBoundaries}
```

In the rest of document we use the outlier boundaries found with the more conservative identifier `SPLUSQuartileIdentifierParameters`.

```
In[114]:= p =
```

```
p ⇒
```

```
ERTMonFindVariableOutlierBoundaries[SPLUSQuartileIdentifierParameters] ⇒ ERTMonEchoValue;
```

» value: <| Humidity → {0.374, 1.07}, Pressure → {1005.95, 1029.55}, Temperature → {-6.03, 43.97}, WindSpeed → {-5.085, 28.235} |>

Conversion of event records to time series

The grouped event records are converted into time series with the function `ERTMonEntityVariableGroupsToTimeSeries`. The time series are aligned to a time point specification given as an argument. The argument can be: a date object, “MinTime”, “MaxTime”, or “None”. (“MaxTime” is the default.)

```
In[115]:= p ⇒
```

```
ERTMonEntityVariableGroupsToTimeSeries["MinTime"] ⇒
```

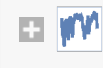


```
ERTMonEchoFunctionContext[#timeSeries[1, 3, 5] &];
```

» <| {KMFL1, Temperature} → TimeSeries[ Time: 0 to 9 417 600 Data points: 110], {KMFL1, Humidity} → TimeSeries[ Time: 0 to 9 417 600 Data points: 110], {KTPA1, Temperature} → TimeSeries[ Time: 0 to 9 417 600 Data points: 110] |>

Compare the last output with the output of the following command.

```

In[116]:= p =
  p ⇒
  ERTMonEntityVariableGroupsToTimeSeries["MaxTime"] ⇒
  ERTMonEchoFunctionContext[#timeSeries[1, 3, 5]] &;

» { {KMFL1, Temperature} → TimeSeries[ Time: -9417600 to 0 Data points: 110], {KMFL1, Humidity} → TimeSeries[ Time: -9417600 to 0 Data points: 110], {KTPA1, Temperature} → TimeSeries[ Time: -9417600 to 0 Data points: 110] }

```

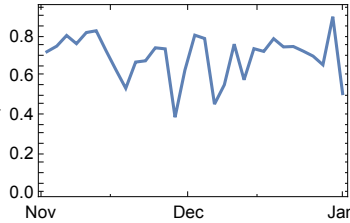
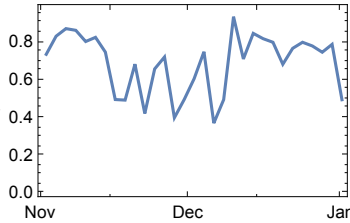
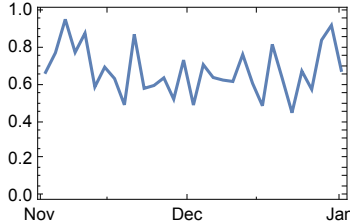
Time series restriction and aggregation.

The main goal of ERTMon is to convert a diverse, general collection of event records into a collection of aligned time series over specified regular time grids.

The regular time grids are specified with the columns “MaxHistoryLength” and “AggregationIntervalLength” of the computation specification. The time series of the variables in the computation specification are restricted to the corresponding maximum history lengths and are aggregated using the corresponding aggregation lengths and functions.

```

In[117]:= p =
  p ⇒
  ERTMonAggregateTimeSeries ⇒
  ERTMonEchoFunctionContext[DateListPlot /@ #timeSeries[1, 3, 5]] &;

» { {KMFL1, Humidity.Mean} → , {KNIP1, Humidity.Mean} → , {KMDW1, Humidity.Mean} →  }

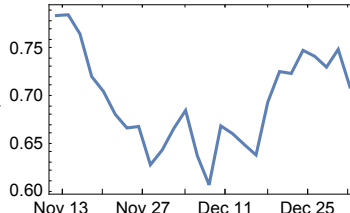
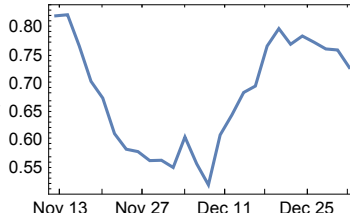

```

Application of time series functions

At this point we can apply time series modifying functions. An often used such function is moving average.

```

In[118]:= p ⇒
  ERTMonApplyTimeSeriesFunction[MovingAverage[#, 6] &] ⇒
  ERTMonEchoFunctionValue[DateListPlot /@ #[[1, 3, 5]] &;

» { {KMFL1, Humidity.Mean} → , {KNIP1, Humidity.Mean} → , {KMDW1, Humidity.Mean} →  }

```

Note that the result is given as a pipeline value, the value of the context key “timeSeries” is not changed.

(In the future, the computation specification and its handling might be extended to handle moving average or other time series function specifications.)

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.

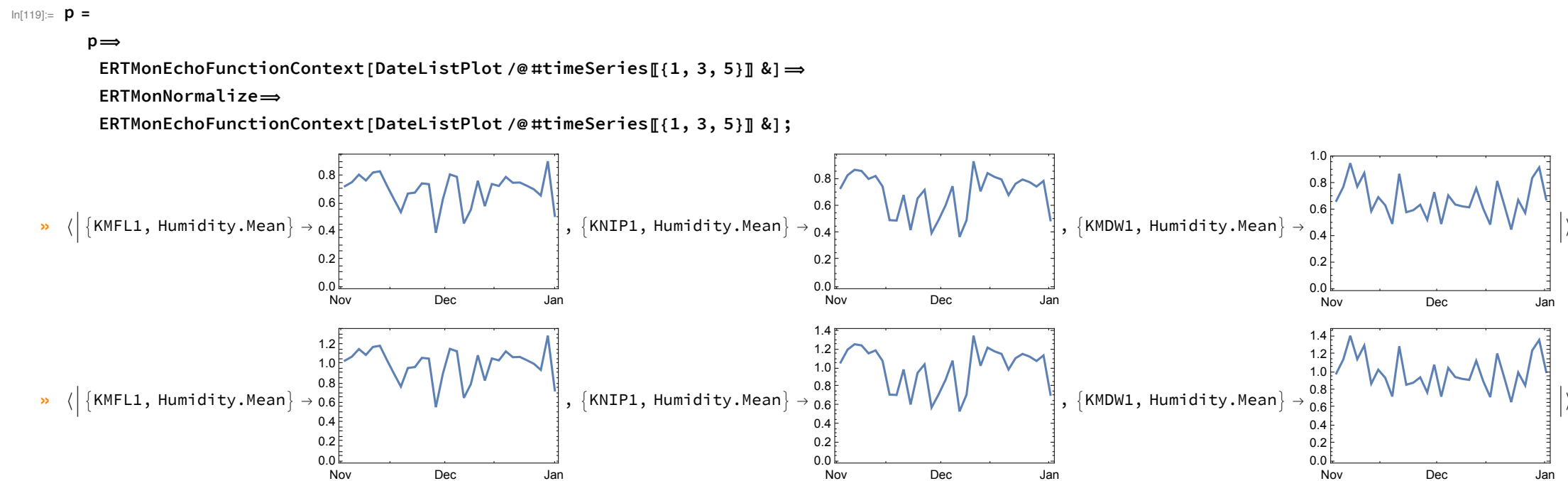
Normalization is done at three different scopes: “entity”, “attribute”, and “variable”.

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.

The following command demonstrates the normalization effect -- compare the y-axes scales of the time series corresponding to the same entity-variable pair.



Here are the normalization values that should be used when normalizing “unseen data.”

```
In[120]:= p ==> ERTMonTakeNormalizationValues
Out[120]:= < | { Humidity.Range, Country, USA } -> 0.0960115, { Humidity.Range, Country, UK } -> 0.0701049, { Pressure.Range, Country, USA } -> 2.86607,
{ Temperature.Range, Country, USA } -> 2.25512, { Temperature.Range, Country, UK } -> 1.48557, { WindSpeed.Range, Country, USA } -> 4.83525, { WindSpeed.Range, Country, UK } -> 4.89506 |>
```

Making contingency matrices

One of the main goals of ERTMon is to produce contingency matrices corresponding to the event records data.

The contingency matrices are created and stored as `SSparseMatrix` objects, [AAp7].

```
In[121]:= p =
p ==> ERTMonMakeContingencyMatrices;
```

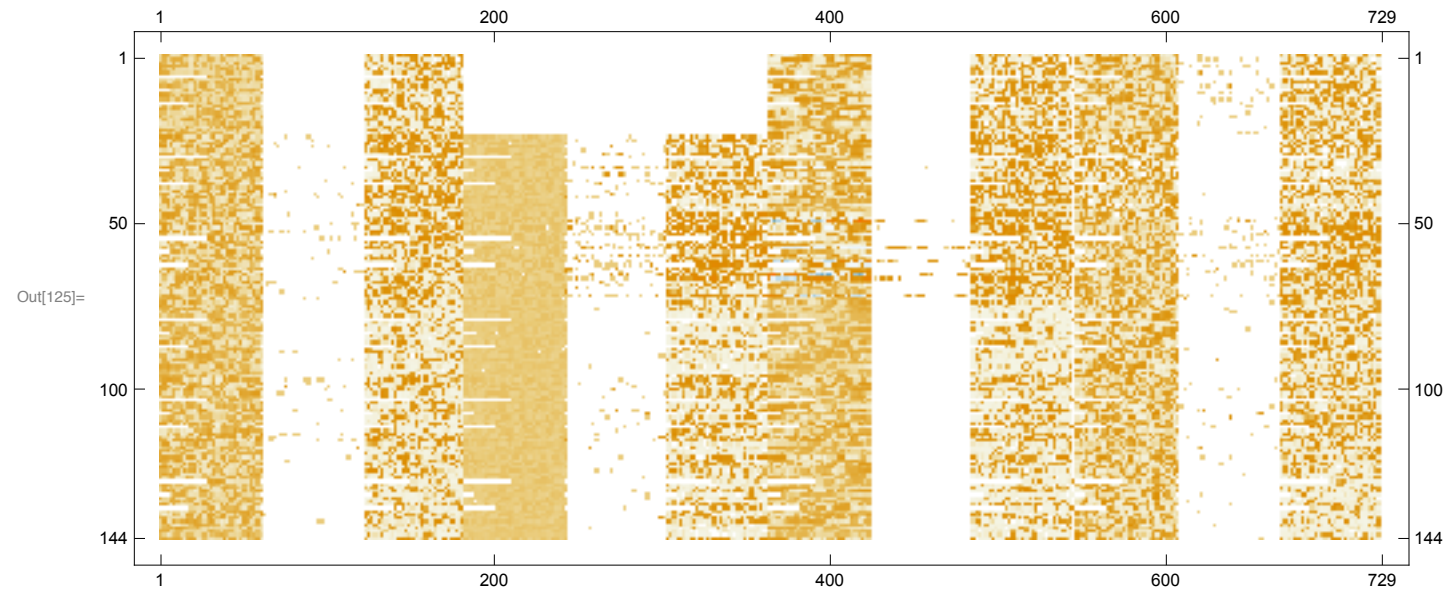
We can obtain an association of the contingency matrices for each variable-and-aggregation-function pair, or obtain the overall contingency matrix.

```
In[122]:= p => ERTMonTakeContingencyMatrices
Dimensions /@ %
```

```
Out[122]:= { { Humidity.Mean -> SparseArray[ + [Specified elements: 4289, Dimensions: {144, 61}], Humidity.OutliersCount -> SparseArray[ + [Specified elements: 100, Dimensions: {144, 61}], Humidity.Range -> SparseArray[ + [Specified elements: 4107, Dimensions: {144, 61}],
Pressure.Mean -> SparseArray[ + [Specified elements: 3563, Dimensions: {144, 60}], Pressure.OutliersCount -> SparseArray[ + [Specified elements: 262, Dimensions: {144, 60}], Pressure.Range -> SparseArray[ + [Specified elements: 3213, Dimensions: {144, 60}],
Temperature.Mean -> SparseArray[ + [Specified elements: 4289, Dimensions: {144, 61}], Temperature.OutliersCount -> SparseArray[ + [Specified elements: 60, Dimensions: {144, 61}], Temperature.Range -> SparseArray[ + [Specified elements: 4069, Dimensions: {144, 61}],
WindSpeed.Mean -> SparseArray[ + [Specified elements: 4249, Dimensions: {144, 61}], WindSpeed.OutliersCount -> SparseArray[ + [Specified elements: 188, Dimensions: {144, 61}], WindSpeed.Range -> SparseArray[ + [Specified elements: 4055, Dimensions: {144, 61}]} ] }
```

```
Out[123]:= { { Humidity.Mean -> {144, 61}, Humidity.OutliersCount -> {144, 61}, Humidity.Range -> {144, 61}, Pressure.Mean -> {144, 60},
Pressure.OutliersCount -> {144, 60}, Pressure.Range -> {144, 60}, Temperature.Mean -> {144, 61}, Temperature.OutliersCount -> {144, 61},
Temperature.Range -> {144, 61}, WindSpeed.Mean -> {144, 61}, WindSpeed.OutliersCount -> {144, 61}, WindSpeed.Range -> {144, 61} } }
```

```
In[124]:= smat = p => ERTMonTakeContingencyMatrix;
MatrixPlot[smat, ImageSize -> 700]
```



```
In[126]:= RowNames[smat]
```

```
Out[126]:= { EGLC1, EGLC10, EGLC11, EGLC12, EGLC13, EGLC14, EGLC15, EGLC16, EGLC17, EGLC18, EGLC19, EGLC2, EGLC20, EGLC21, EGLC22, EGLC23, EGLC24, EGLC3, EGLC4, EGLC5, EGLC6, EGLC7, EGLC8, EGLC9, KFTY1,
KFTY10, KFTY11, KFTY12, KFTY13, KFTY14, KFTY15, KFTY16, KFTY17, KFTY18, KFTY19, KFTY2, KFTY20, KFTY21, KFTY22, KFTY23, KFTY24, KFTY3, KFTY4, KFTY5, KFTY6, KFTY7, KFTY8, KFTY9, KMDW1,
KMDW10, KMDW11, KMDW12, KMDW13, KMDW14, KMDW15, KMDW16, KMDW17, KMDW18, KMDW19, KMDW2, KMDW20, KMDW21, KMDW22, KMDW23, KMDW24, KMDW3, KMDW4, KMDW5, KMDW6, KMDW7, KMDW8, KMDW9, KMFL1,
KMFL10, KMFL11, KMFL12, KMFL13, KMFL14, KMFL15, KMFL16, KMFL17, KMFL18, KMFL19, KMFL2, KMFL20, KMFL21, KMFL22, KMFL23, KMFL24, KMFL3, KMFL4, KMFL5, KMFL6, KMFL7, KMFL8, KMFL9, KNIP1,
KNIP10, KNIP11, KNIP12, KNIP13, KNIP14, KNIP15, KNIP16, KNIP17, KNIP18, KNIP19, KNIP2, KNIP20, KNIP21, KNIP22, KNIP23, KNIP24, KNIP3, KNIP4, KNIP5, KNIP6, KNIP7, KNIP8, KNIP9, KTPA1,
KTPA10, KTPA11, KTPA12, KTPA13, KTPA14, KTPA15, KTPA16, KTPA17, KTPA18, KTPA19, KTPA2, KTPA20, KTPA21, KTPA22, KTPA23, KTPA24, KTPA3, KTPA4, KTPA5, KTPA6, KTPA7, KTPA8, KTPA9 }
```

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[] =>
 ERTMonSetEventRecords[eventRecords] =>
 ERTMonSetEntityAttributes[entityAttributes] =>
 ERTMonEchoDataSummary =>
 ERTMonSetComputationSpecification[compSpec] =>
 ERTMonGroupEntityVariableRecords =>
 ERTMonComputeVariableStatistic[Histogram[#1, ImageSize -> Small] &] =>
 ERTMonEchoFunctionValue["Variable distributions:", #1 &] =>
 ERTMonFindVariableOutlierBoundaries[SPLUSQuartileIdentifierParameters] =>
 ERTMonEchoFunctionValue["Outlier boundaries:", #1 &] =>
 ERTMonEntityVariableGroupsToTimeSeries["MaxTime"] =>
 ERTMonAggregateTimeSeries =>

 ERTMonMakeContingencyMatrices =>
 ERTMonEchoFunctionValue["Contingency matrices:", (MatrixPlot[ToSSparseMatrix[#1], ImageSize -> Small] &) /@ #1 &]

initialize the monad pipeline
ingest event records
ingest event attributes
show data summary
ingest computation specification
group records per {EntityID, Variable}
find variable distributions
echo found distributions
find outliers boundaries for each variable
echo found outliers boundaries
align the event records and create time series
aggregate the time series
(according to the computation specification)
make contingency matrices
plot contingency matrices

» Data summary: <|eventRecords → {

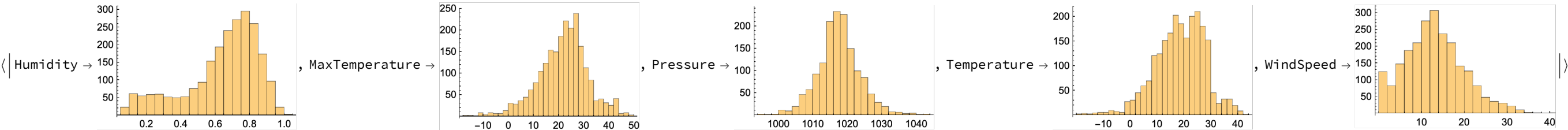
1 EntityID	2 LocationID	3 ObservationTime	4 Variable	5 Value
KNIP	1967	Jacksonville	1967	Min 3 689 539 200
WM095866	1965	Melbourne	1965	1st Qu 3 705 955 200
KMDW	1802	Chicago	1802	Mean 3.71474×10^9
KMFL	1798	Miami	1798	Median 3 715 113 600
KGEU	1572	Peoria	1572	3rd Qu 3 723 235 200
EGLC	1456	London	1456	Max 3 739 392 000

}, entityAttributes → {

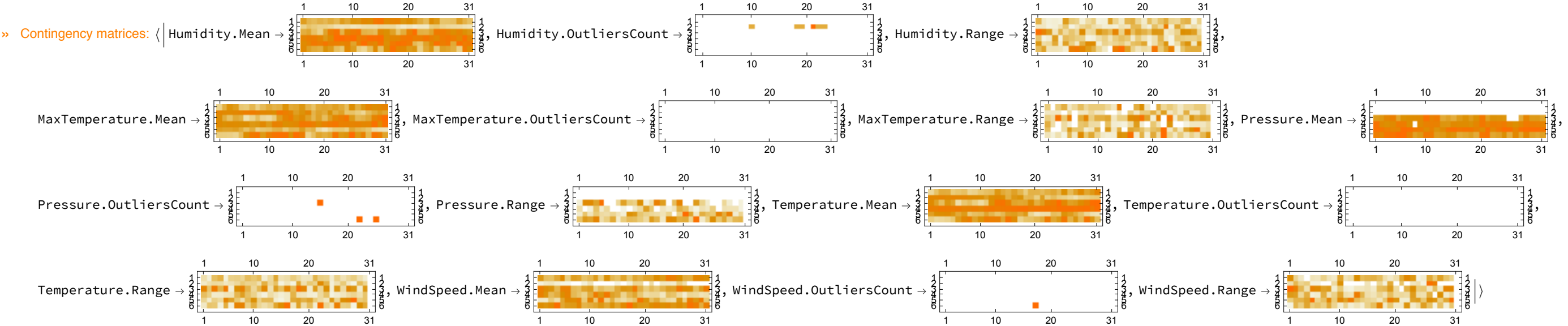
1 EntityID	2 Attribute	3 Value
EGLC	2	USA 4
KGEU	2	Australia 1
KMDW	2	Chicago 1
KMFL	2	Jacksonville 1
KNIP	2	London 1
WM095866	2	Melbourne 1
		(Other) 3

>>>

» Variable distributions:



» Outlier boundaries: <| Humidity → {0.176, 1.168}, MaxTemperature → {-1.67, 45.45}, Pressure → {1003.75, 1031.35}, Temperature → {-5.805, 43.755}, WindSpeed → {-5.005, 30.555} |>



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, Monad code generation and extension, (2017), MathematicaForPrediction at WordPress.

URL: <https://mathematicaforprediction.wordpress.com/2017/06/23/monad-code-generation-and-extension> .

[AA1a] Anton Antonov, Monad code generation and extension, (2017), MathematicaForPrediction at GitHub.

[AA2] Anton Antonov, A monad for classification workflows, (2018), MathematicaForPrediction at WordPress.

URL: <https://mathematicaforprediction.wordpress.com/2018/05/15/a-monad-for-classification-workflows> .