Introduction

The objective of the library is to convert input data into standard data sample objects. The data sample objects can then be transformed and converted into a proper format for display and statistical analysis.

As an illustration of the library, let's consider an example of data inputs for panel data analysis. The panel data observation typically consists of a date, when the observation was measured, id of the entity related to the observation, and id of the variable related to the observation. The dates represent the time series component of the data, entities ids represent the cross-section component of the data and variables ids represent the multi-variable component of the data. Overall, the data can be viewed as a three-dimensional set of observations, where dimensions are represented by dates, entity ids, and variable ids.

The multi-dimensional set of data observations is modelled using a tree structure, where each dimension is represented by a respective set of tree keys at a given tree level. In the panel data example, the data set is modelled using a three-level tree with the dates representing the first-level keys, entity ids representing the second-level keys, and variable ids representing the third-level keys.

The tree structure can be transformed to provide different ways of data representation. The tree structure can also be converted to other formats used in statistical analysis. Similarly, the sample tree structure can be used as an intermediary stage for conversion of other objects into the data formats used in statistical analysis.

Sample object

Sample object structure

A standard sample object is modelled as a tree

```
key_1 => {key_2 => … … {key_n => value}}
```

where each observation is described by a collection of keys and a value. A value is either a number or a class that implements ISampleValue interface, which may include other fields in addition to the numeric value.

Standard examples include the following sample tree structures:

- Time series: `variable id => {date => value}`
- Cross-section regression: `variable id => {observation index => value}`
- Panel data: `variable id => {date => {observation index => value}}`

Sample Factory

Sample inputs are generally described by a two-dimensional array with multiple column keys and multiple headers which also describe sample keys. In an illustrative example below, sample keys are represented by two columns [Date, ID] and one header with the variable name. The factory class will create a three-level sample tree of the form `variable id => {Date => {ID => value}}}`.

Date	ID	INDUSTRY	UNEMP1	UNEMP2	UNEMP3	UNEMP4
31-Mar-92	20	2	7.00	7.10	7.05	7.00
31-Mar-92	30	3	7.00	7.20	7.20	7.20
31-Mar-92	35	2	7.00	7.10	7.10	7.20
31-Mar-92	40	2	7.00	7.10	7.10	7.00
31-Mar-92	60	2	7.00	7.00	6.90	6.70
31-Mar-92	62	2	6.90	7.10	7.00	6.80
31-Mar-92	65	1	7.00	7.10	6.90	6.70
31-Mar-92	84	1	7.00	7.20	7.40	7.50
31-Mar-92	94	1	7.00	7.20	7.20	6.90
31-Mar-92	99	1	7.00	7.10	6.90	6.80

The tree structure can be set using a configuration list (referred to as **`key-ranking`**) with the following format

```
{c0, r0, r1}
```

which specifies the order in which the keys path to each sample value is created. Note that the above example also represents a default format if no key-order list is specified in the sample factory class. The default order is use first data headers as key and then use keys described in respective data columns.

In addition to the key order, the sample factory class needs the data types as inputs. If data types are not specified, the factory class assumes string type by default. In the above example, the key type mapping (referred to as **`key-types`**) is described as follows:

```
r0 => date

r1 => int

c0 => string
```

The date type is presented either by Excel date numeric type or parsed from a string. An int or double type is either presented by a numeric object or parsed from respective string. Specifying the types is important to ensure that the data returned back to Excel is in correct format (date types are not converted and returned back to Excel as strings).

Sample Transformations

The package implements a set of various sample transformations listed below.

- **Key Swap**. A sample is created with a specific order of sample keys. In certain cases the order of keys needs to be modified. This is implemented using a sequence of key swap transformation in which the order of two neighbor keys is swapped.
- **Sample copy**. The transformation creates a new copy of the sample.

Sample Converter

In certain cases the tree structure of the sample needs to be converted into a two-dimensional array. He array is created into two stages:

- Conversion to two-dimensional **mapping**. For a certain sample tree threshold level, the keys with levels less or equal to the threshold level are used as the first key path and the keys with levels greater than the threshold level are used as the second key path of the mapping.
- Conversion to two-dimensional **list**. A two-dimensional mapping is converted than to a list. In a list structure, a set of all second-level key path is created. If row `i` is represented by a first-level key-path and column `j` is represented by a second-level key-path, then the element at position `(i, j)` is null, if the element (level-one key-path, level-two key-path) does not exist, and equal to the respective element if the element exists.

Forecast Sample

A forecast sample is a specific sample with three-level tree structure

```
variable id => {date => {forecaster id => IForecast}}
```

where IForecast object extends ISampleValue interface. In addition to the forecast numeric value, the IForecast object contains also the maturity date and tenor fields.

The forecast sample is implemented for the following purpose:

- Creating forecast samples.
- Graphical representation of the forecasts.
- Comparing the forecast values to actual values. Performing linear regression analysis to estimate the relationship between forecasts and actual values. The regression analysis is performed for both the average forecast as well as for each individual professional forecaster.
- Correlation analysis of the forecasts from different professional forecasters (using principal component analysis).

Each element of the forecast sample analysis is discussed below.

Forecast Sample Factory

Forecast sample inputs are generally represented in two formats:

- **Format A**. The first format is a standard format of panel data that was illustrated above. The two key columns describe the forecast issue date and forecaster id keys; the columns headers include the variable names. Standard key order is applied to create the `variable id => {date => {forecaster id => IForecast}}` structure. In addition, the mapping `variable id => forecast tenor` is used as inputs. Each forecast maturity date is estimated based on the issued date and the forecast tenor.

  The configuration mappings of the forecast sample are described as follows:

  `key-types` mapping:

  ```
  r0 => issue-date
  r1 => forecaster-id
  c1 => variable-id
  ```

  `variable-to-maturity` mapping

  variable name => maturity of the respective forecast

  Note that the sample configuration mappings are not used directly as inputs in the forecast sample factory class but instead derived from the `key-types` mapping. By default, the ranking of the forecast sample keys is the following: `(variable-id, issue-date, forecaster-id)`.

The maturity date is derived from the issue date and the variable name that corresponds to a specific tenor specified in the `variable-to-maturity` mapping.

- **Format B**. In the alternative format, the data is presented as follows. Column headers include the variables ids and forecasters' ids. A single key column describes the forecast issue date. In addition, the mapping `variable id => forecast maturity date` is used as inputs. Each forecast tenor is estimated based on the issued date and the forecast maturity date.

In the first case, the tenors are typically round numbers typically equal to one, two, three, four quarters, etc. In the second case, the forecast tenors can often be arbitrary numbers and not equal exactly to a specific numbers of quarters. Multiple forecasts can be published for a specific quarter and the forecasts can be published both prior to and after the quarter start.

In the second case, the sample is filtered to retain only the forecasts closest to specific forecast tenor equal to a whole quarters' number of forecast maturity horizon. After the filtering, the forecast samples under the two formats become essentially the same and the same tools are applicable to both formats.

Forecast Sample Graphical Representation

Forecast sample is generally represented in three graphical formats:

1. Actual and forecasted values (with fixed tenors) at the forecast issue date. The data in this format is the input data. The input data is aggregated into statistics described by the sample of the following structure: `variable id => issue date => forecast[value = statistic value]`. The sample is then converted into array, which is displayed by respective graph;
2. Actual and forecasted values (with fixed tenors) at the forecast maturity date. The output generated above is swapped into the following sample: `variable id => maturity date => forecast[value = statistic value]`. The sample is then converted into array, which is displayed by respective graph;
3. Actual values and forecasted values with different tenors (at the forecasts' issue date). The sample created in step 1 is swapped into the following sample: `issue date => variable id => forecast[value = statistic value]`. The `variable-id` columns correspond to different maturity horizons of the forecast sample. The sample is converted into a two dimensional array ;

The three output formats are illustrated below.

Fixed tenor forecasts at the issue date

Fixed tenor forecasts at the maturity date

Forecast term structure

Forecast Sample Regression Analysis

Forecasts Correlation Analysis

Modelling

Sample object are modelled using java framework and can be created and used either using Excel or jmc functions interfaces.

Java

Java package `jmathkr.lib.stats.basic.sample` models Sample objects frameworks.

Excel

Sample functions are called through Excel using `AC.stats.sample` function reference. The Excel interface for java sample frameworks is implemented in `jmathkr.lib.server.xlloop.functions.stats.basic` package via the following functions:

Jmc