Orange3-Timeseries Documentation

-

Sep 29, 2021

Contents

CHAPTER 1

Widgets

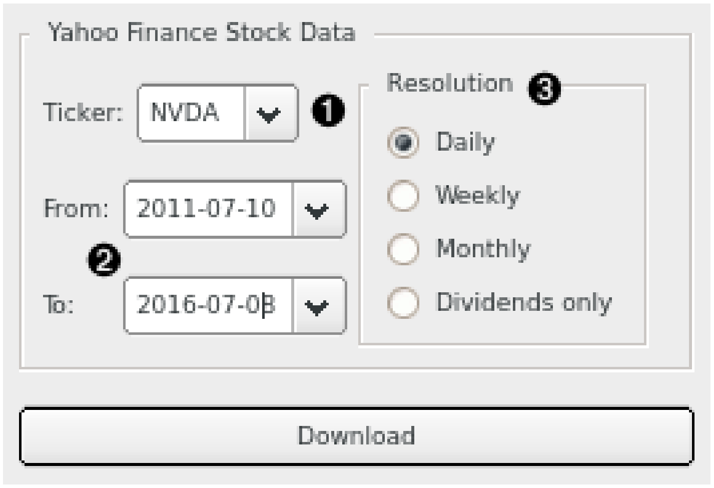
1.1 Yahoo Finance

Generate time series from Yahoo Finance stock market data.

Outputs

- Time series: Time series table of open, high, low, close (OHLC) prices, volume and adjusted close price.

This widget fetches historical stock market data from Yahoo Finance and outputs it as a time series data table.



Yahoo Finance Stock Data

Ticker: NVDA ❶

From: 2011-07-10

❷

To: 2016-07-08

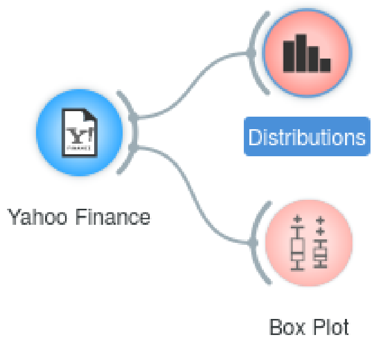
Resolution ❸

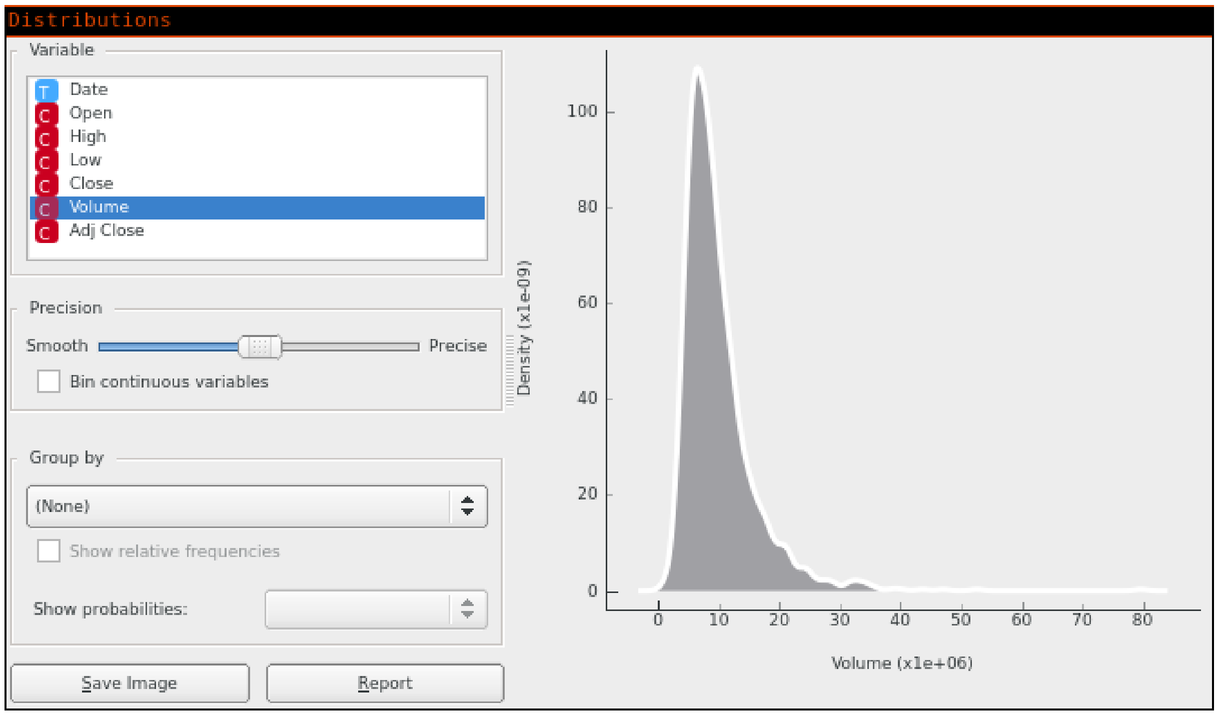
[✓ Daily] Weekly Monthly Dividends only

1. Stock (e.g. GOOG) or index (e.g. DJI) symbol you are interested in.
2. Date range you are interested in.
3. Desired resolution of the time series. Can be one of: daily, weekly, monthly, or dividends. The last option outputs a table of dates when dividends were issued, along with their respective dividend amounts.

1.1.1 Example

Since the output data type is inherently a **Table**, you can connect it to wherever a data table is expected. Naturally, you can use it to test some functions in the Timeseries add-on.





1.2 As Timeseries

Reinterpret a **Table** object as a Timeseries object.

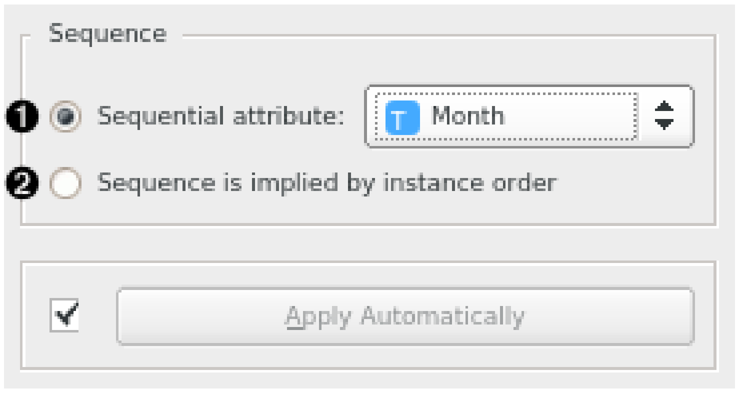
Inputs

- Data: Any data table.

Outputs

- Time series: Data table reinterpreted as time series.

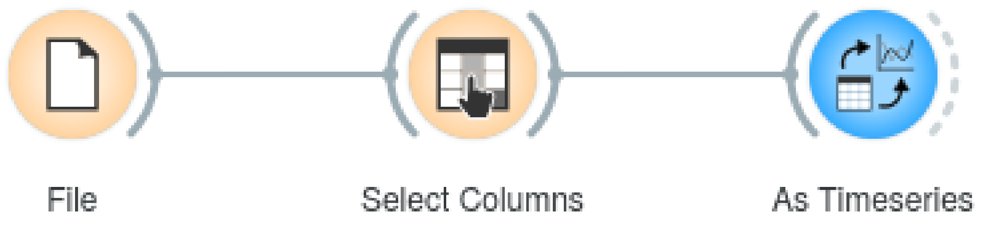
This widget reinterprets any data table as a time series, so it can be used with the rest of the widgets in this add-on. In the widget, you can set which data attribute represents the time variable.



1. The time attribute, the values of which imply measurements' order and spacing. This can be any continuous attribute.
2. Alternatively, you can specify that the time series sequence is implied by instance order.

1.2.1 Example

The input to this widget comes from any data-emitting widget, e.g. the **File** widget. Note, whenever you do some processing with Orange core widgets, like the **Select Columns** widget, you need to re-apply the conversion into time series.



1.3 Interpolate

Induce missing values in the time series by interpolation.

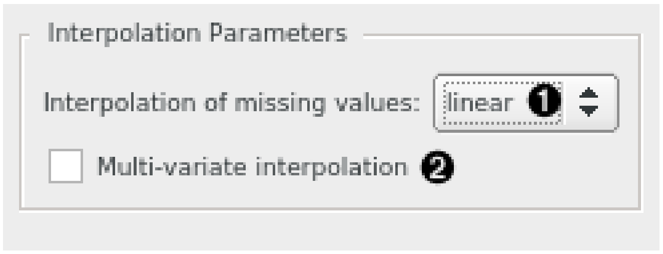
Inputs

- Time series: Time series as output by **As Timeseries** widget.

Outputs

- Time series: The input time series with the chosen default interpolation method for when the algorithms require interpolated time series (without missing values).
- Interpolated time series: The input time series with any missing values interpolated according to the chosen interpolation method.

Most time series algorithms assume, you don't have any missing values in your data. In this widget, you can chose the interpolation method to impute the missing values with. By default, it's linear interpolation (fast and reasonable default).

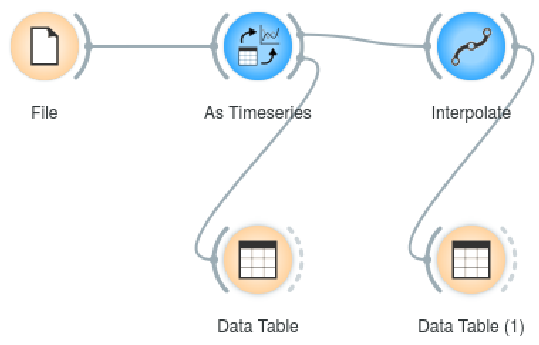

Interpolation Parameters
Interpolation of missing values: linear 1
Multi-variate interpolation 2

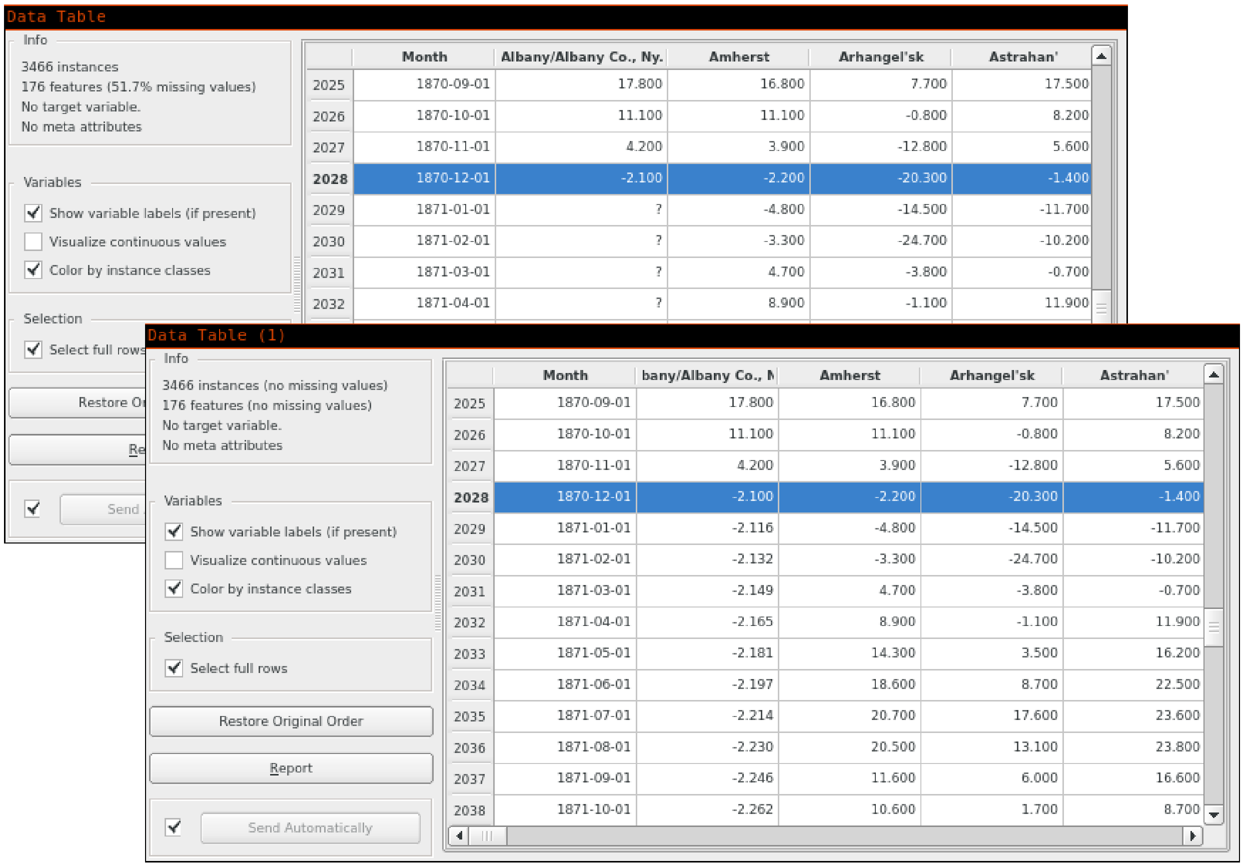
1. Interpolation type. You can select one of linear, cubic spline, nearest, or mean interpolation:
   - **Linear** interpolation replaces missing values with linearly-spaced values between the two nearest defined data points.
   - **Spline** interpolation fits a cubic polynomial to the points around the missing values. This is a painfully slow method that usually gives best results.
   - **Nearest** interpolation replaces missing values with the previous defined value.
   - **Mean** interpolation replaces missing values with the series' mean.
2. **Multi-variate interpolation** interpolates the whole series table as a two-dimensional plane instead of as separate single-dimensional series.

Missing values on the series' end points (head and tail) are always interpolated using *nearest* method. Unless the interpolation method is set to *nearest*, discrete time series (i.e. sequences) are always imputed with the series' *mode* (most frequent value).

1.3.1 Example

Pass a time series with missing values in, get interpolated time series out.





Info

3466 instances
176 features (51.7% missing values)
No target variable.
No meta attributes

Variables

- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

	Month	Albany/Albany Co., Ny.	Amherst	Arhangel'sk	Astrahan'
2025	1870-09-01	17.800	16.800	7.700	17.500
2026	1870-10-01	11.100	11.100	-0.800	8.200
2027	1870-11-01	4.200	3.900	-12.800	5.600
2028	1870-12-01	-2.100	-2.200	-20.300	-1.400
2029	1871-01-01	?	-4.800	-14.500	-11.700
2030	1871-02-01	?	-3.300	-24.700	-10.200
2031	1871-03-01	?	4.700	-3.800	-0.700
2032	1871-04-01	?	8.900	-1.100	11.900

Info

3466 instances (no missing values)
176 features (no missing values)
No target variable.
No meta attributes

Variables

- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

	Month	bany/Albany Co., N	Amherst	Arhangel'sk	Astrahan'
2025	1870-09-01	17.800	16.800	7.700	17.500
2026	1870-10-01	11.100	11.100	-0.800	8.200
2027	1870-11-01	4.200	3.900	-12.800	5.600
2028	1870-12-01	-2.100	-2.200	-20.300	-1.400
2029	1871-01-01	-2.116	-4.800	-14.500	-11.700
2030	1871-02-01	-2.132	-3.300	-24.700	-10.200
2031	1871-03-01	-2.149	4.700	-3.800	-0.700
2032	1871-04-01	-2.165	8.900	-1.100	11.900
2033	1871-05-01	-2.181	14.300	3.500	16.200
2034	1871-06-01	-2.197	18.600	8.700	22.500
2035	1871-07-01	-2.214	20.700	17.600	23.600
2036	1871-08-01	-2.230	20.500	13.100	23.800
2037	1871-09-01	-2.246	11.600	6.000	16.600
2038	1871-10-01	-2.262	10.600	1.700	8.700

1.4 Moving Transform

Apply rolling window functions to the time series. Use this widget to get a series' mean.

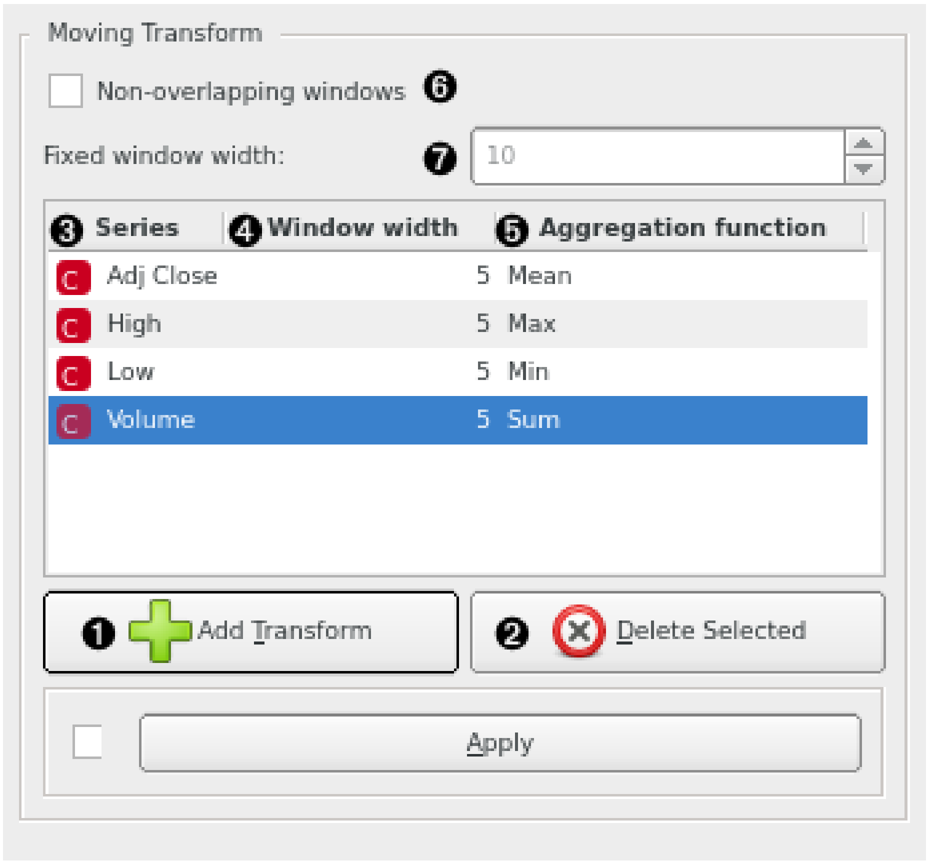
Inputs

- Time series: Time series as output by As Timeseries widget.

Outputs

- Time series: The input time series with the added series' transformations.

In this widget, you define what aggregation functions to run over the time series and with what window sizes.



Non-overlapping windows: ☐

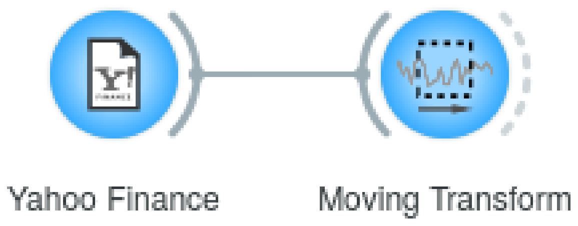
Fixed window width: 10

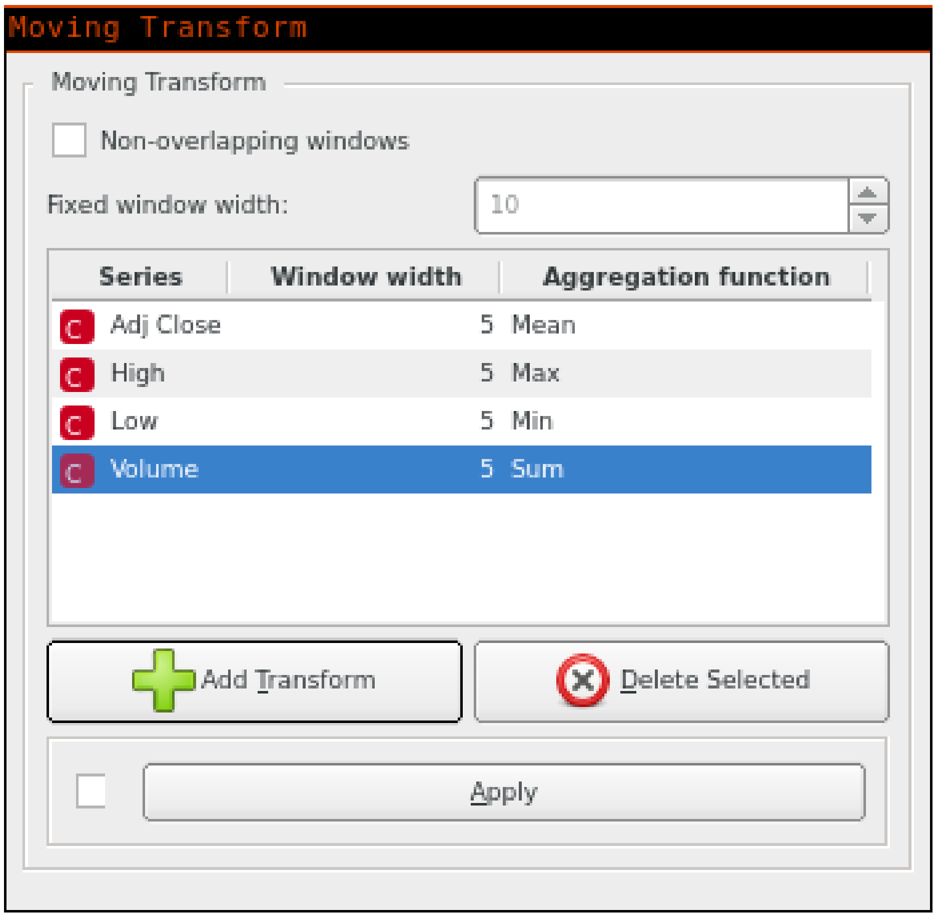
3 Series	4 Window width	5 Aggregation function
C Adj Close	5	Mean
C High	5	Max
C Low	5	Min
C Volume	5	Sum

1. Define a new transformation.
2. Remove the selected transformation.
3. Time series you want to run the transformation over.
4. Desired window size.
5. Aggregation function to aggregate the values in the window with. Options are: *mean*, *sum*, *max*, *min*, *median*, *mode*, *standard deviation*, *variance*, *product*, *linearly-weighted moving average*, *exponential moving average*, *harmonic mean*, *geometric mean*, *non-zero count*, *cumulative sum*, and *cumulative product*.
6. Select *Non-overlapping windows* options if you don't want the moving windows to overlap but instead be placed side-to-side with zero intersection.
7. In the case of non-overlapping windows, define the fixed window width(overrides and widths set in (4).

1.4.1 Example

To get a 5-day moving average, we can use a rolling window with *mean* aggregation.




Moving Transform

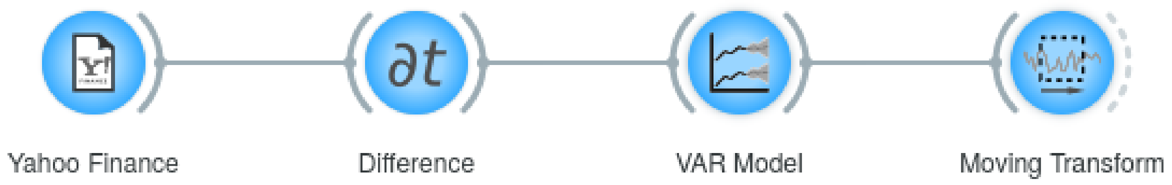
Moving Transform

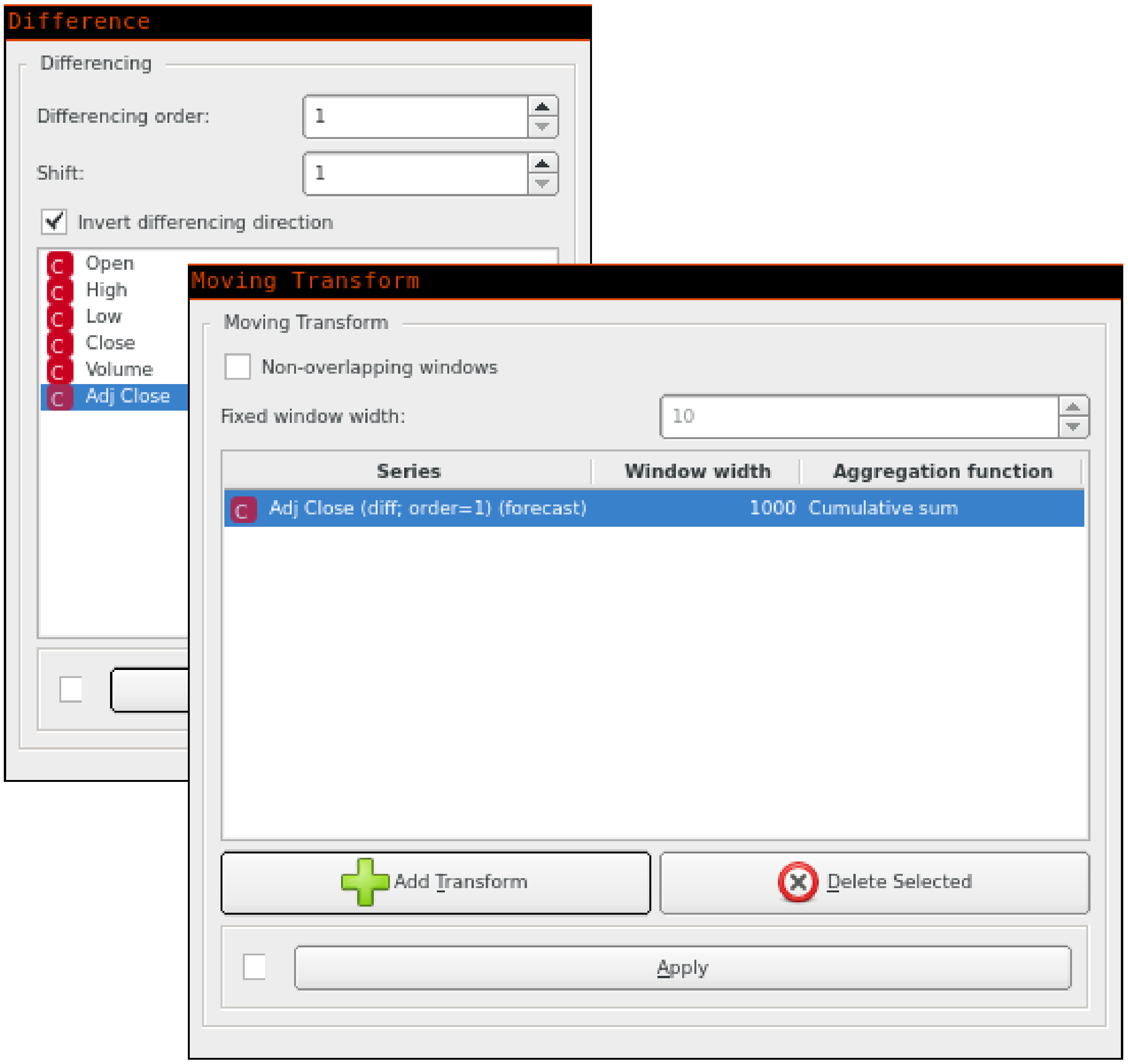
☐ Non-overlapping windows

Fixed window width: 10

Series	Window width	Aggregation function
Adj Close	5	Mean
High	5	Max
Low	5	Min
Volume	5	Sum

To integrate time series' differences from Difference widget, use *Cumulative sum* aggregation over a window wide enough to grasp the whole series.





Difference

Differencing

Differencing order: 1

Shift: 1

[✓] Invert differencing direction

- Open
- High
- Low
- Close
- Volume
- Adj Close

Moving Transform

Moving Transform

[ ] Non-overlapping windows

Fixed window width: 10

Series	Window width	Aggregation function
Adj Close (diff; order=1) (forecast)	1000	Cumulative sum

See also

Seasonal Adjustment

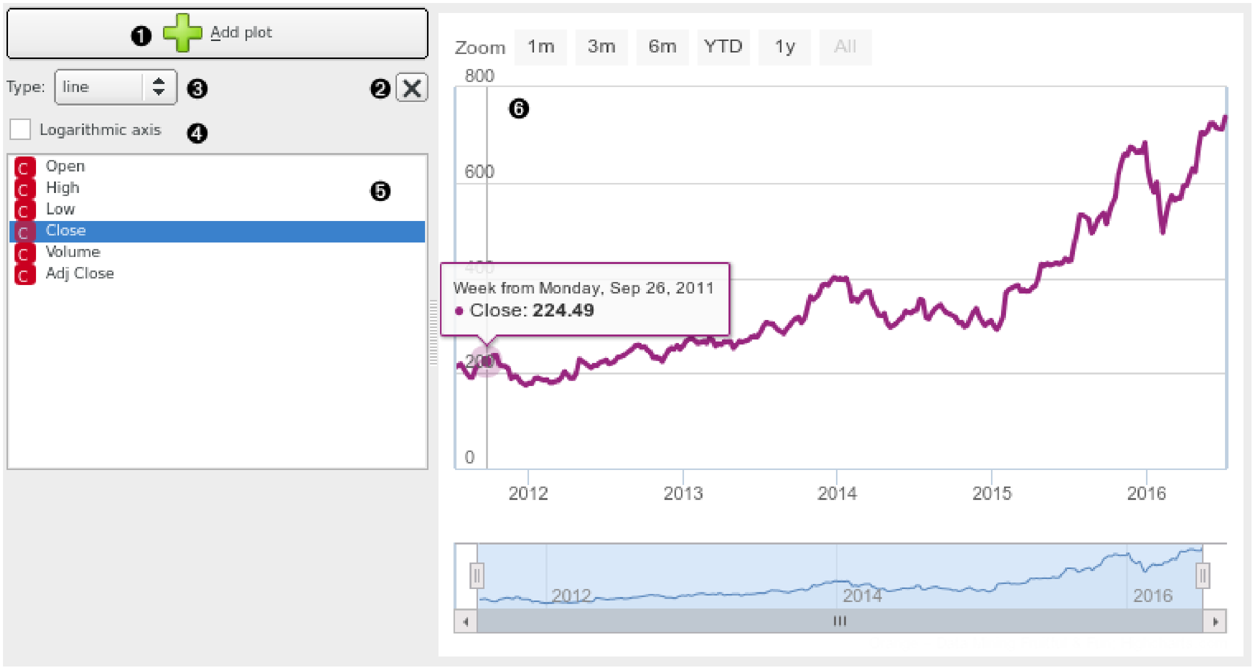
1.5 Line Chart

Visualize time series' sequence and progression in the most basic time series visualization imaginable.

Inputs

- Time series: Time series as output by As Timeseries widget.
- Forecast: Time series forecast as output by one of the models (like VAR or ARIMA).

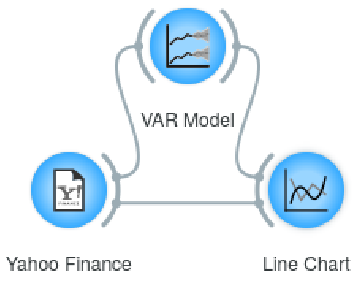
You can visualize the time series in this widget.

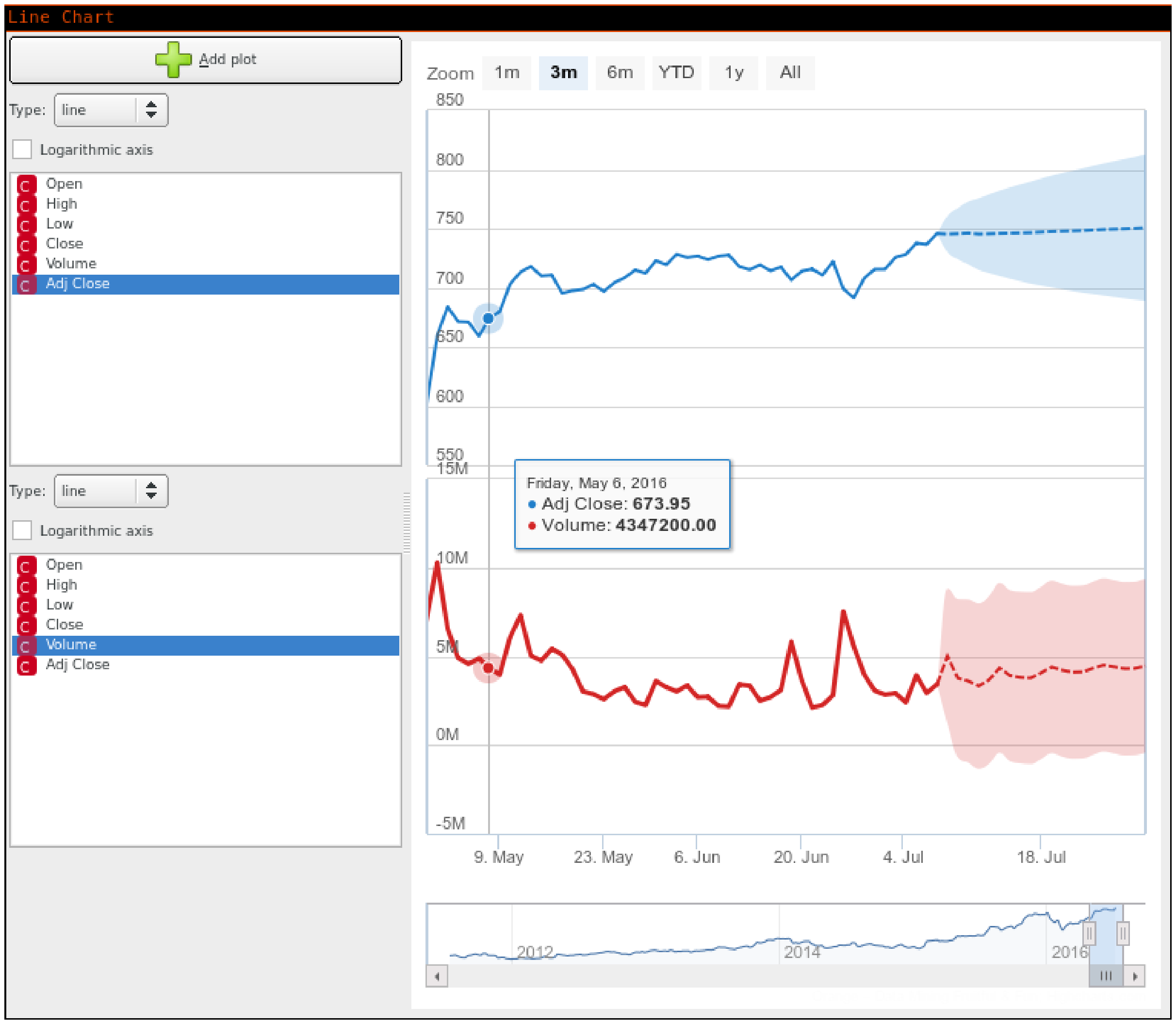


1. Stack a new line chart below the current charts.
2. Remove the associated stacked chart.
3. Type of chart to draw. Options are: *line*, *step line*, *column*, *area*, *spline*.
4. Switch between linear and logarithmic *y* axis.
5. Select the time series to preview (select multiple series using the *Ctrl* key).
6. See the selected series in this area.

1.5.1 Example

Attach the model's forecast to the *Forecast* input signal to preview it. The forecast is drawn with a dotted line and the confidence intervals as an ranged area.





Friday, May 6, 2016
● Adj Close: **673.95**
● Volume: **4347200.00**

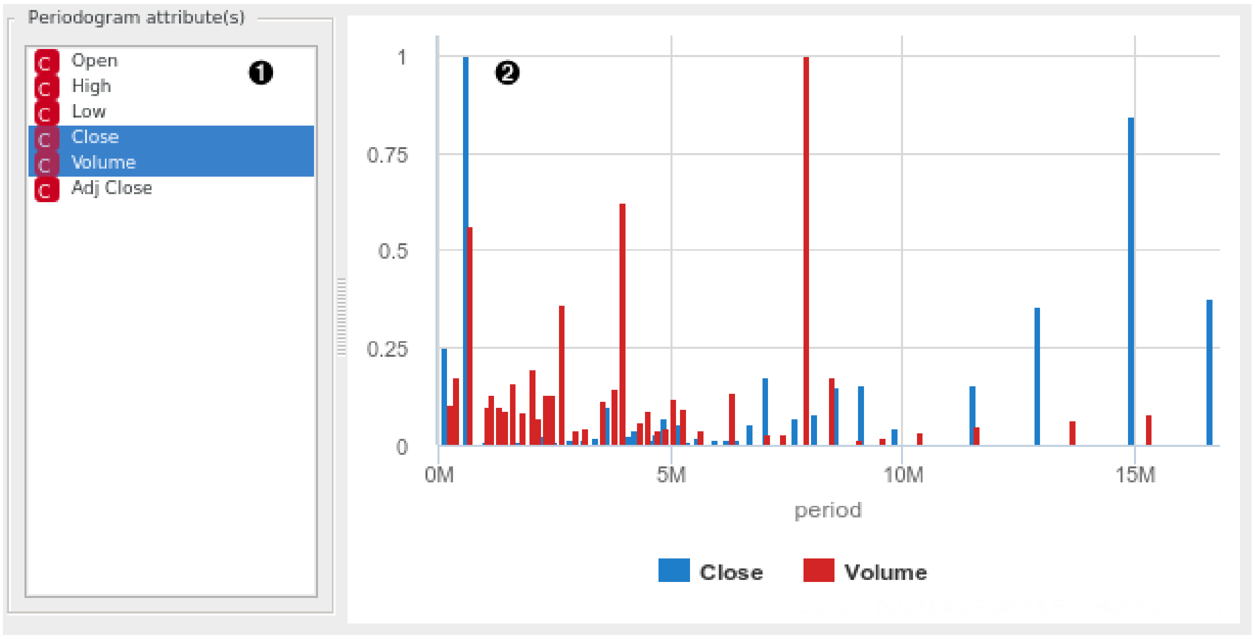
1.6 Periodogram

Visualize time series' cycles, seasonality, periodicity, and most significant periods.

Inputs

- Time series: Time series as output by As Timeseries widget.

In this widget, you can visualize the most significant periods of the time series.



1. Select the series to calculate the periodogram for.
2. See the periods and their respective relative power spectral density estimates.

Periodogram for non-equispaced series is calculated using Lomb-Scargle method.

1.6.1 See also

Correlogram

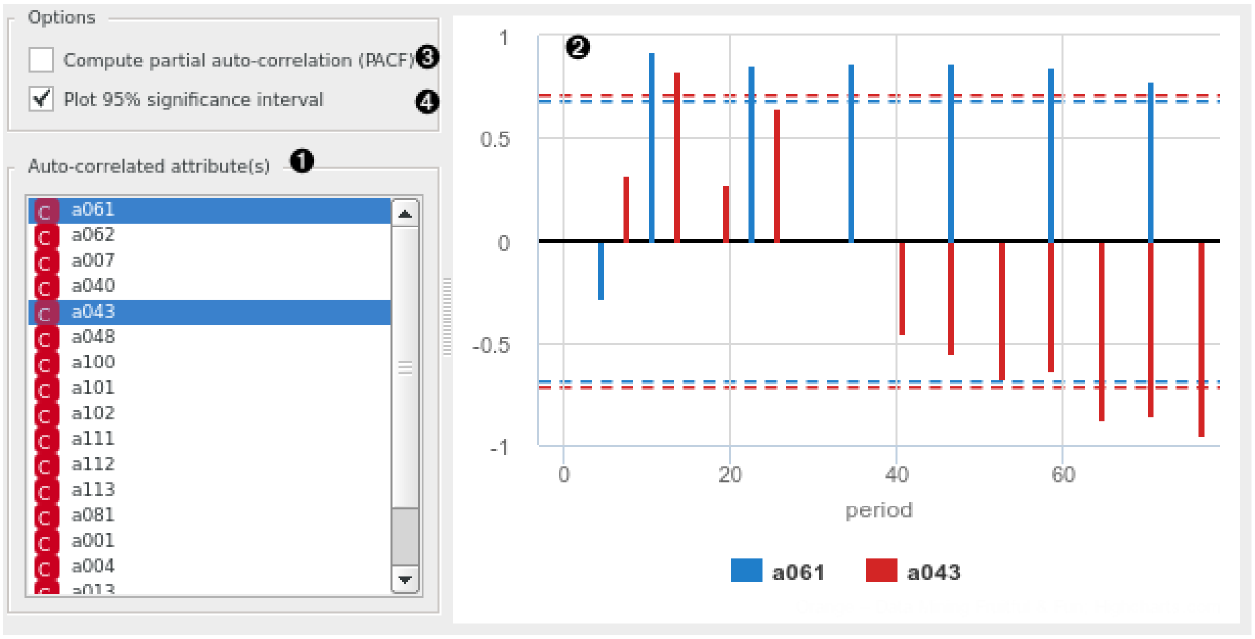
1.7 Correlogram

Visualize variables' auto-correlation.

Inputs

- Time series: Time series as output by As Timeseries widget.

In this widget, you can visualize the autocorrelation coefficients for the selected time series.


Options
Compute partial auto-correlation (PACF) 3
✓ Plot 95% significance interval 4
Auto-correlated attribute(s) 1
a061
a062
a007
a040
a043
a048
a100
a101
a102
a111
a112
a113
a081
a001
a004
a013
2
1
0.5
0
-0.5
-1
0
20
40
60
period
a061
a043

1. Select the series to calculate autocorrelation for.
2. See the autocorrelation coefficients.
3. Choose to calculate the coefficients using partial autocorrelation function (PACF) instead.
4. Choose to plot the 95% significance interval (dotted horizontal line). Coefficients that are outside of this interval might be significant.

1.7.1 See also

Periodogram

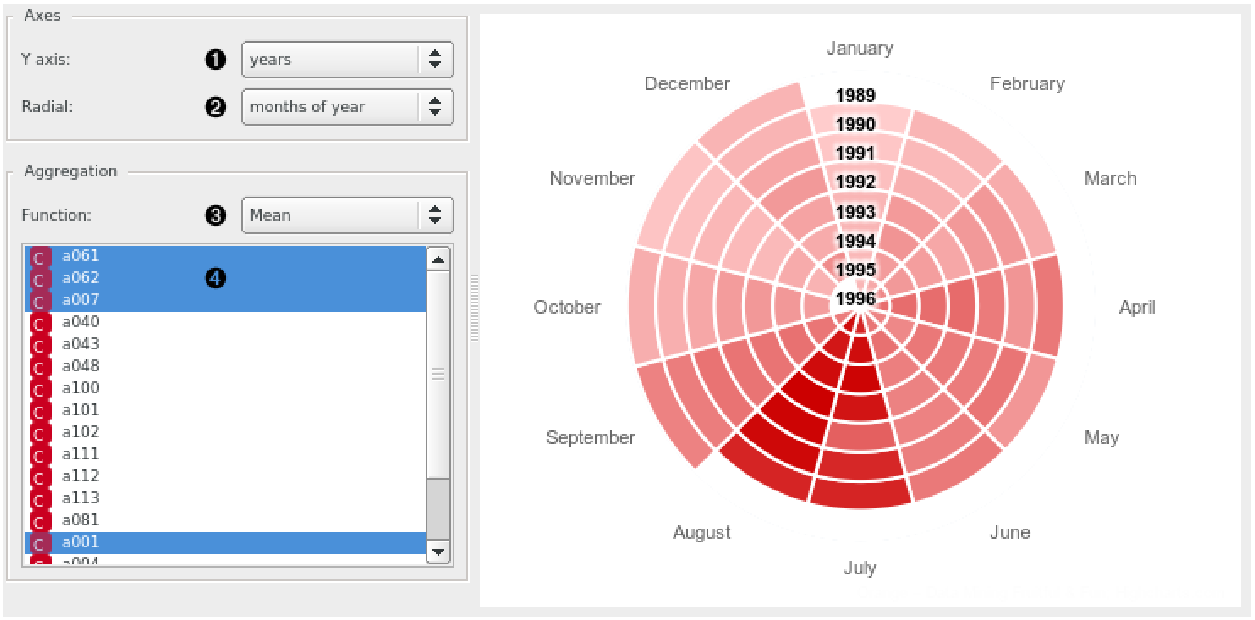
1.8 Spiralogram

Visualize variables' auto-correlation.

Inputs

- Time series: Time series as output by As Timeseries widget.

Visualize time series' periodicity in a spiral heatmap.


Axes
Y axis: years
Radial: months of year
Aggregation
Function: Mean

1. Unit of the vertical axis. Options are: years, months, or days (as present in the series), months of year, days of week, days of month, days of year, weeks of year, weeks of month, hours of day, minutes of hour.
2. Unit of the radial axis (options are the same as for (1)).
3. Aggregation function. The series is aggregated on intervals selected in (1) and (2).
4. Select the series to include.

1.8.1 Example

The image above shows traffic for select French highways. We see a strong seasonal pattern (high summer) and somewhat of an anomaly on July 1992. In this month, there was an important trucker strike in protest of the new road laws.

See also

Aggregate

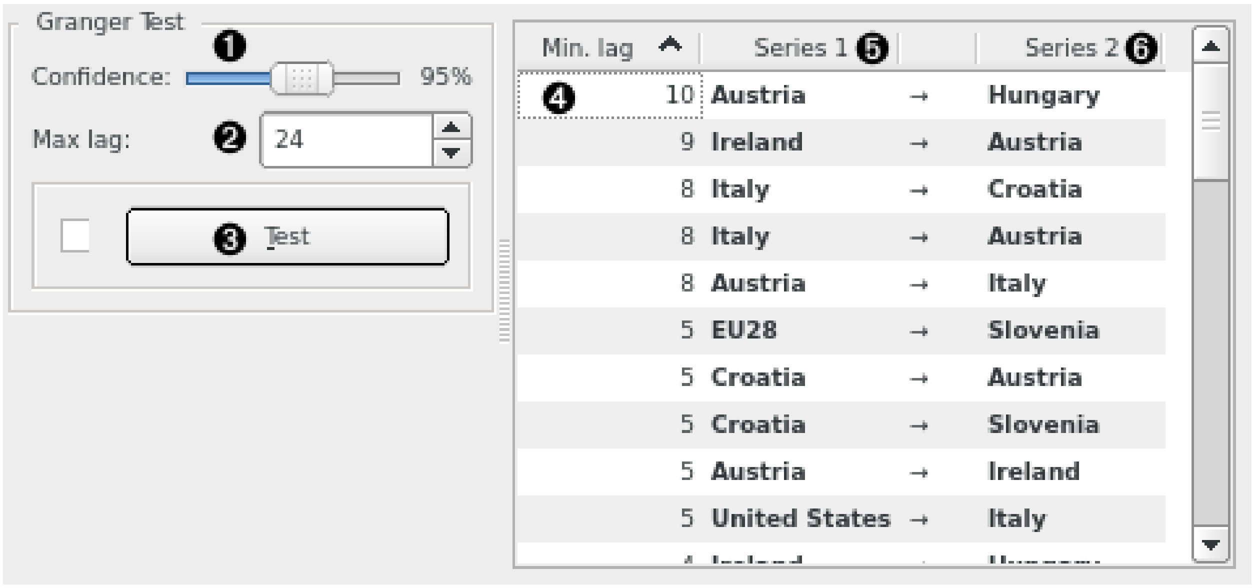
1.9 Granger Causality

Test if one time series Granger-causes (i.e. can be an indicator of) another time series.

Inputs

- Time series: Time series as output by As Timeseries widget.

This widgets performs a series of statistical tests to determine the series that cause other series so we can use the former to forecast the latter.



Confidence: 95%

Max lag: 24

Min. lag	Series 1		Series 2
10	Austria	→	Hungary
9	Ireland	→	Austria
8	Italy	→	Croatia
8	Italy	→	Austria
8	Austria	→	Italy
5	EU28	→	Slovenia
5	Croatia	→	Austria
5	Croatia	→	Slovenia
5	Austria	→	Ireland
5	United States	→	Italy

1. Desired level of confidence.
2. Maximum lag to test to.
3. Runs the test.
4. Denotes the minimum lag at which one series can be said to cause another. In the first line of the example above, if we have the monthly unemployment rate time series for Austria, we can say something about unemployment rate in Hungary 10 months ahead.
5. The causing (antecedent) series.
6. The effect (consequent) series.

The time series that Granger-cause the series you are interested in are good candidates to have in the same VAR model. But careful, even if one series is said to Granger-cause another, this doesn't mean there really exists a causal relationship. Mind your conclusions.

1.10 ARIMA Model

Model the time series using ARMA, ARIMA, or ARIMAX model.

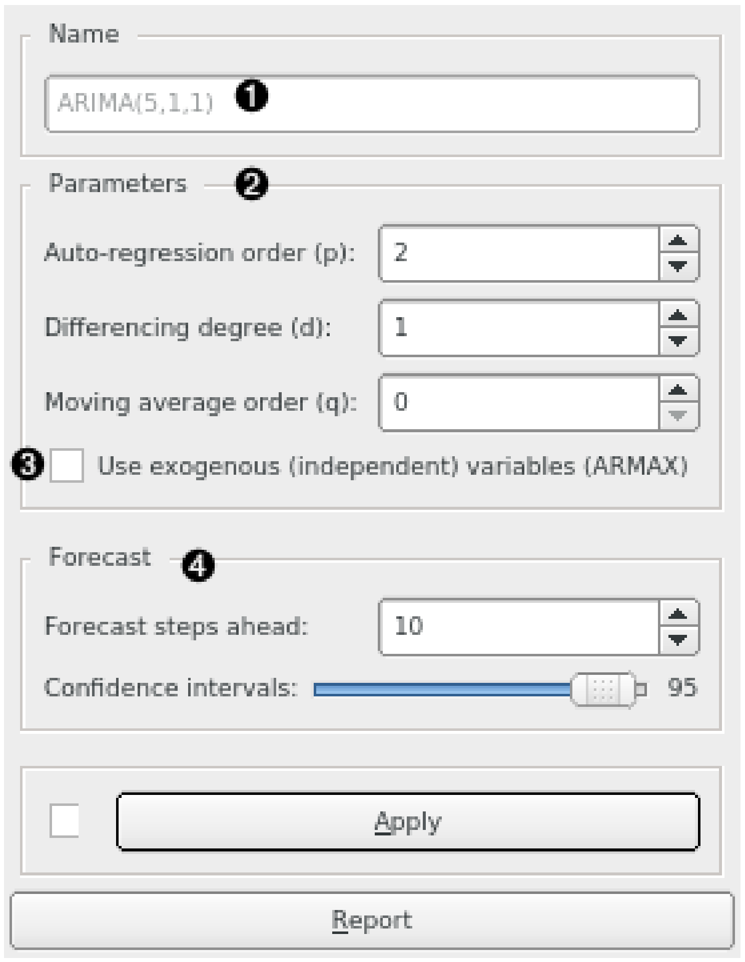
Inputs

- Time series: Time series as output by As Timeseries widget.
- Exogenous data: Time series of additional independent variables that can be used in an ARIMAX model.

Outputs

- Time series model: The ARIMA model fitted to input time series.
- Forecast: The forecast time series.
- Fitted values: The values that the model was actually fitted to, equals to *original values - residuals*.
- Residuals: The errors the model made at each step.

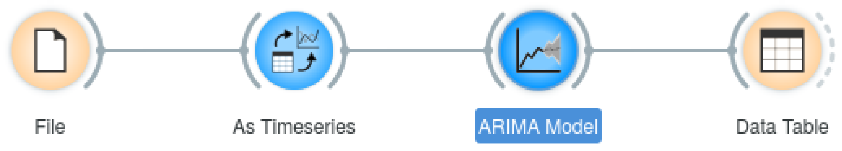
Using this widget, you can model the time series with ARIMA model.

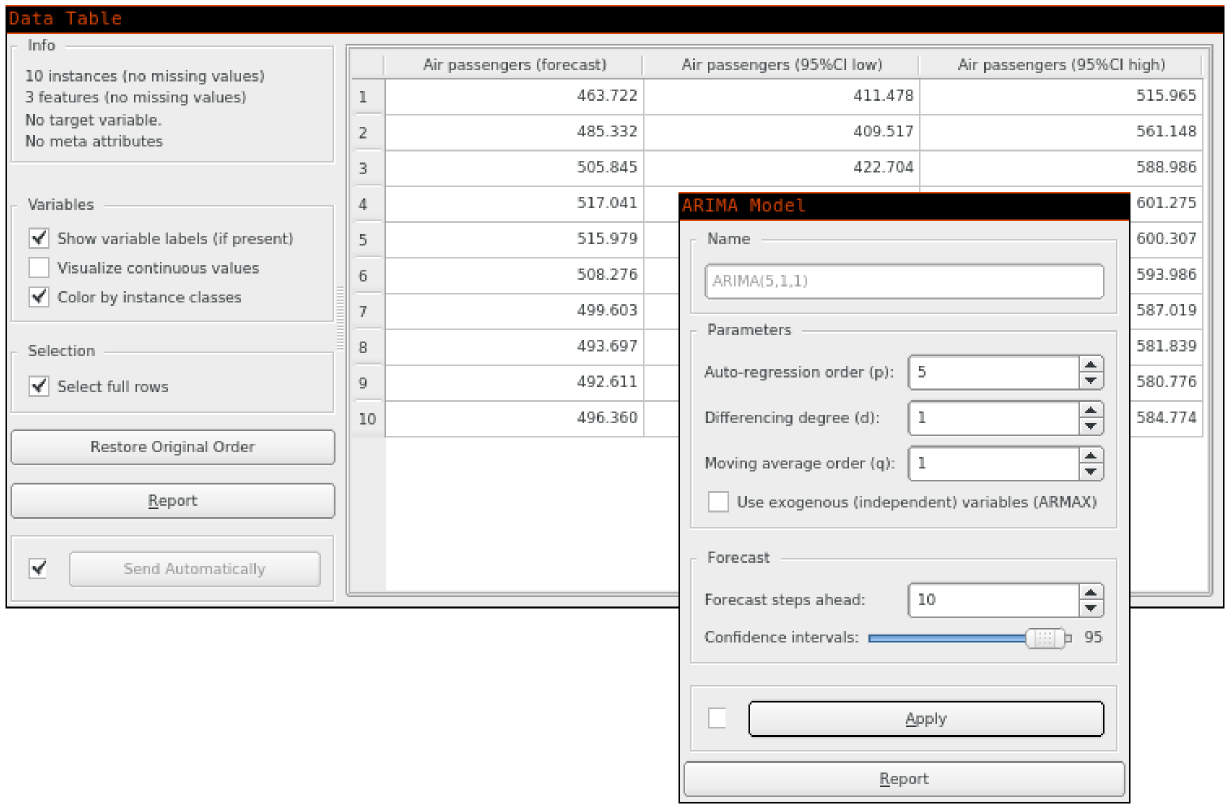


Auto-regression order (p): 2
Differencing degree (d): 1
Moving average order (q): 0
Use exogenous (independent) variables (ARMAX)
Forecast steps ahead: 10
Confidence intervals: 95

1. Model's name. By default, the name is derived from the model and its parameters.
2. ARIMA's p, d, q parameters.
3. Use exogenous data. Using this option, you need to connect additional series on the *Exogenous data* input signal.
4. Number of forecast steps the model should output, along with the desired confidence intervals values at each step.

1.10.1 Example





Info

10 instances (no missing values)
3 features (no missing values)
No target variable.
No meta attributes

Variables

- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

	Air passengers (forecast)	Air passengers (95%CI low)	Air passengers (95%CI high)
1	463.722	411.478	515.965
2	485.332	409.517	561.148
3	505.845	422.704	588.986
4	517.041		601.275
5	515.979		600.307
6	508.276		593.986
7	499.603		587.019
8	493.697		581.839
9	492.611		580.776
10	496.360		584.774

Name: ARIMA(5,1,1)

Parameters

Auto-regression order (p): 5
Differencing degree (d): 1
Moving average order (q): 1
- [ ] Use exogenous (independent) variables (ARMAX)

Forecast

Forecast steps ahead: 10
Confidence intervals: 95

See also

VAR Model, Model Evaluation

1.11 VAR Model

Model the time series using vector autoregression (VAR) model.

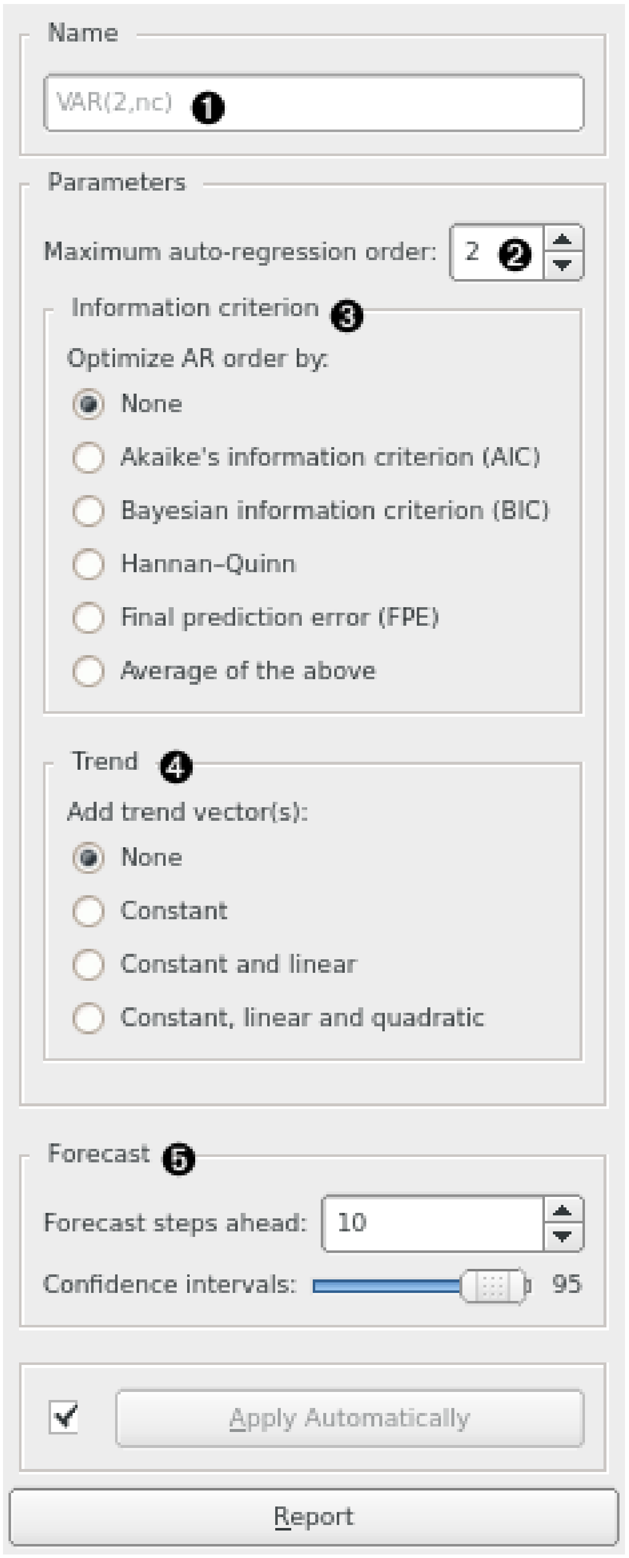
Inputs

- Time series: Time series as output by As Timeseries widget.

Outputs

- Time series model: The VAR model fitted to input time series.
- Forecast: The forecast time series.
- Fitted values: The values that the model was actually fitted to, equals to *original values - residuals*.
- Residuals: The errors the model made at each step.

Using this widget, you can model the time series using VAR model.


Name

VAR(2,nc) 1

Parameters

Maximum auto-regression order: 2 2

Information criterion 3

Optimize AR order by:

(●) None
( ) Akaike's information criterion (AIC)
( ) Bayesian information criterion (BIC)
( ) Hannan–Quinn
( ) Final prediction error (FPE)
( ) Average of the above

Trend 4

Add trend vector(s):

(●) None
( ) Constant
( ) Constant and linear
( ) Constant, linear and quadratic

Forecast 5

Forecast steps ahead: 10

Confidence intervals: 95

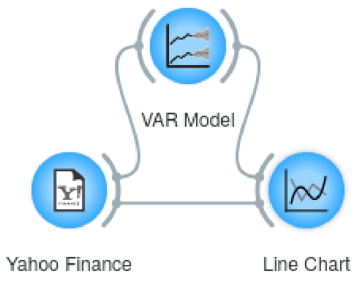
✓ Apply Automatically

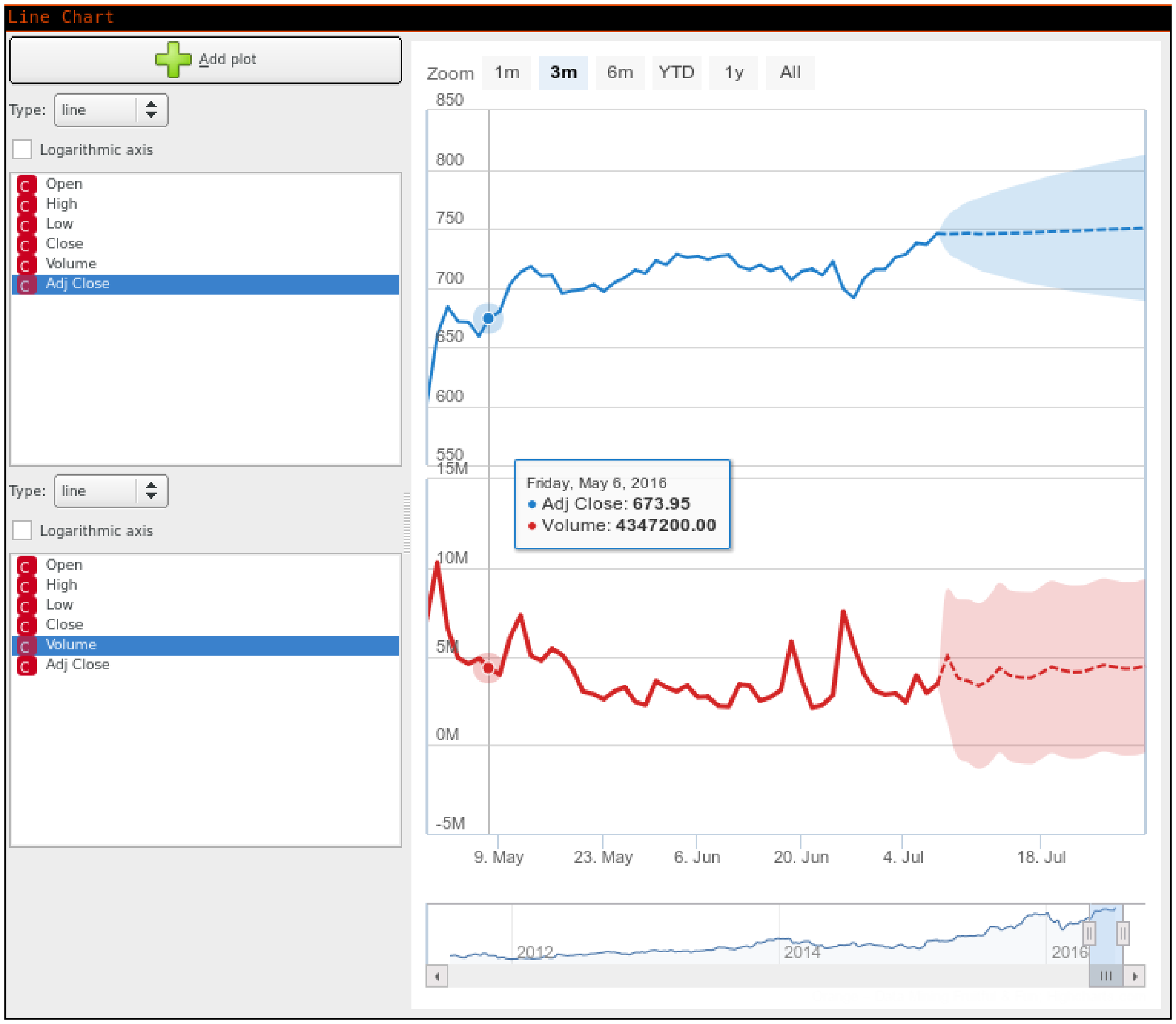
Report

1. Model's name. By default, the name is derived from the model and its parameters.
2. Desired model order (number of parameters).

3. If other than *None*, optimize the number of model parameters (up to the value selected in (2)) with the selected information criterion (one of: AIC, BIC, HQIC, FPE, or a mix thereof).
4. Choose this option to add additional "trend" columns to the data:
    - *Constant*: a single column of ones is added
    - *Constant and linear*: a column of ones and a column of linearly increasing numbers are added
    - *Constant, linear and quadratic*: an additional column of quadratics is added
5. Number of forecast steps the model should output, along with the desired confidence intervals values at each step.

1.11.1 Example





Friday, May 6, 2016
● Adj Close: **673.95**
● Volume: **4347200.00**

See also

ARIMA Model, Model Evaluation

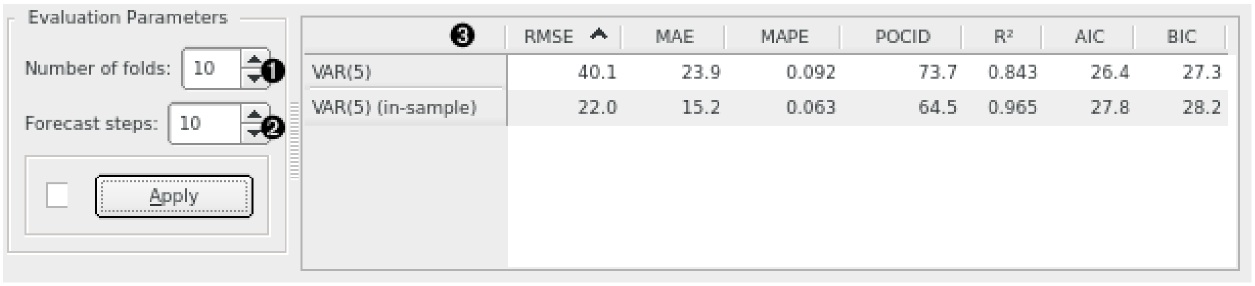
1.12 Model Evaluation

Evaluate different time series' models.

Inputs

- Time series: Time series as output by As Timeseries widget.
- Time series model(s): The time series model(s) to evaluate (e.g. VAR or ARIMA).

Evaluate different time series' models. by comparing the errors they make in terms of: root mean squared error (RMSE), median absolute error (MAE), mean absolute percent error (MAPE), prediction of change in direction (POCID), coefficient of determination (R^2), Akaike information criterion (AIC), and Bayesian information criterion (BIC).



	RMSE	MAE	MAPE	POCID	R²	AIC	BIC
VAR(5)	40.1	23.9	0.092	73.7	0.843	26.4	27.3
VAR(5) (in-sample)	22.0	15.2	0.063	64.5	0.965	27.8	28.2

1. Number of folds for time series cross-validation.
2. Number of forecast steps to produce in each fold.
3. Results for various error measures and information criteria on cross-validated and in-sample data.

This slide (source) shows how cross validation on time series is performed. In this case, the number of folds (1) is 10 and the number of forecast steps in each fold (2) is 1.

In-sample errors are the errors calculated on the training data itself. A stable model is one where in-sample errors and out-of-sample errors don't differ significantly.

####See also

ARIMA Model, VAR Model

1.13 Time Slice

Select a slice of measurements on a time interval.

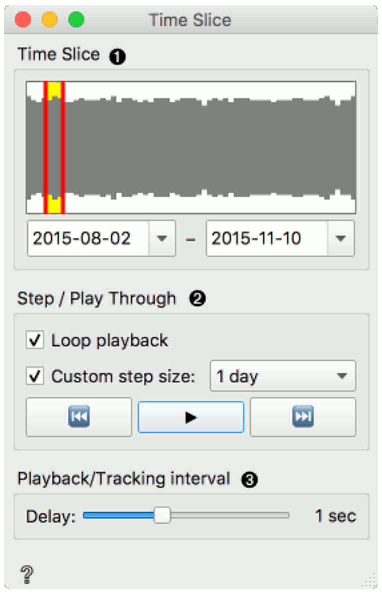
Inputs

- Data: Time series as output by As Timeseries widget.

Outputs

- Subset: Selected time slice from the time series.

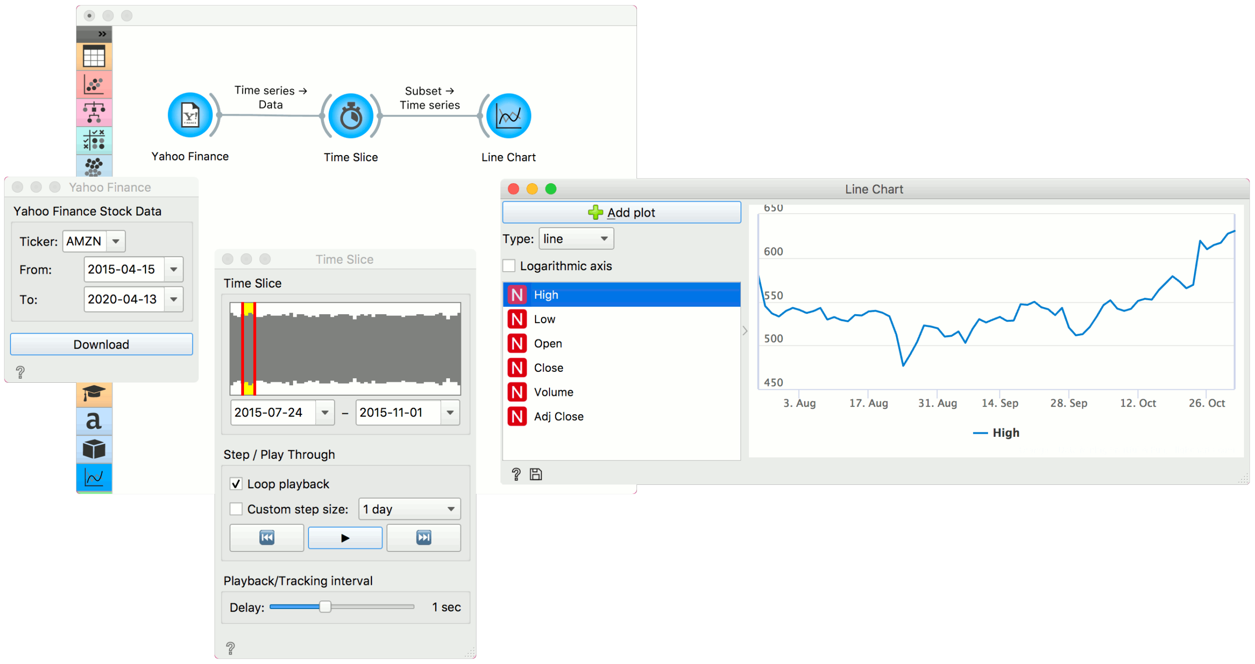
Time slice is a subset selection widget designed specifically for time series and for interactive visualizations. It enables selecting a subset of the data by date and/or hour. Moreover, it can output data from a sliding window with options for step size and speed of the output change.



1. Visual representation of time series with the selected time slice. Click and drag the red lines to adjust the time window, or click and drag the yellow frame to move it around. Alternatively, set the to and from dates below to output the desired subset.
2. If *Loop playback* is selected the data will 'replay' continuously. *Custom step size* defines how the time slice move. If it is set to, say, *1 day*, the window will output n + 1 day once it moves. Without the custom step size defined, the slice will move to the next frame of the same size without any overlap. Press play to being the sliding window and stop to stop it. Backwards and forwards buttons move the slice by the specified step size.
3. Set the speed of the sliding window.

1.13.1 Example

This simple example uses Yahoo Finance widget to retrieve financial data from Yahoo, namely the AMNZ stock index from 2015 to 2020. Next, we will use **Time Slice** to observe how the data changed through time. We can observe the output of Time Slice in Line Chart. Press *Play* in Time Slice and see how Line Chart changes interactively.



1.14 Aggregate

Aggregate data by second, minute, hour, day, week, month, or year.

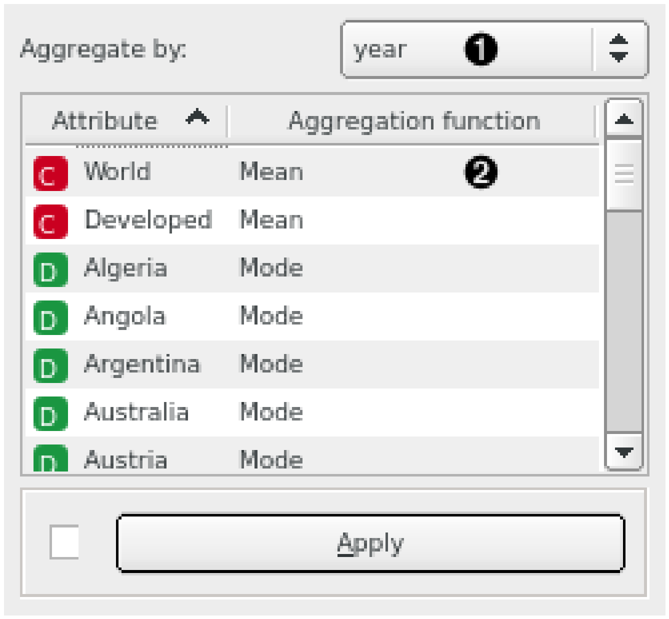
Inputs

- Time series: Time series as output by **As Timeseries** widget.

Outputs

- Time series: Aggregated time series.

Aggregate joins together instances at the same level of granularity. In other words, if aggregating by day, all instances from the same day will be merged into one. Aggregation function can be defined separately based on the type of the attribute.



Attribute	Aggregation function
C World	Mean
C Developed	Mean
D Algeria	Mode
D Angola	Mode
D Argentina	Mode
D Australia	Mode
D Austria	Mode

1. Interval to aggregate the time series by. Options are: second, minute, hour, day, week, month, or year.
2. Aggregation function for each of the time series in the table. Discrete variables (sequences) can only be aggregated using mode (i.e. most frequent value), whereas string variables can only be aggregated using string concatenation.

1.14.1 See also

Moving Transform

1.15 Difference

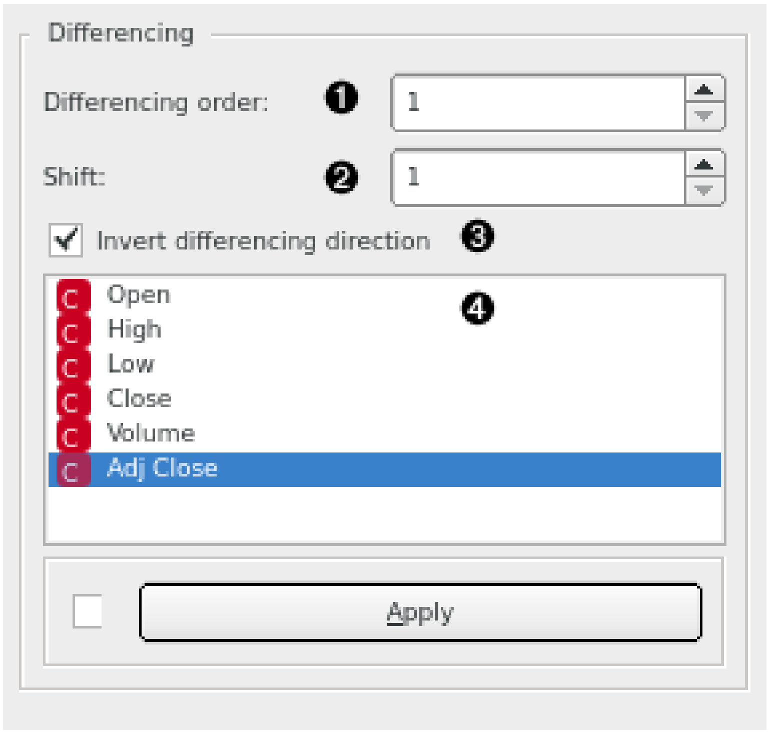
Make the time series stationary by replacing it with 1st or 2nd order discrete difference along its values.

Inputs

- Time series: Time series as output by As Timeseries widget.

Outputs

- Time series: Differences of input time series.



1. Order of differencing. Can be 1 or 2.
2. The shift before differencing. Value of 1 equals to discrete differencing. You can use higher values to compute the difference between now and this many steps ahead.
3. Invert the differencing direction.
4. Select the series to difference.

To integrate the differences back into the original series (e.g. the forecasts), use the *Moving Transform* widget.

1.16 Seasonal Adjustment

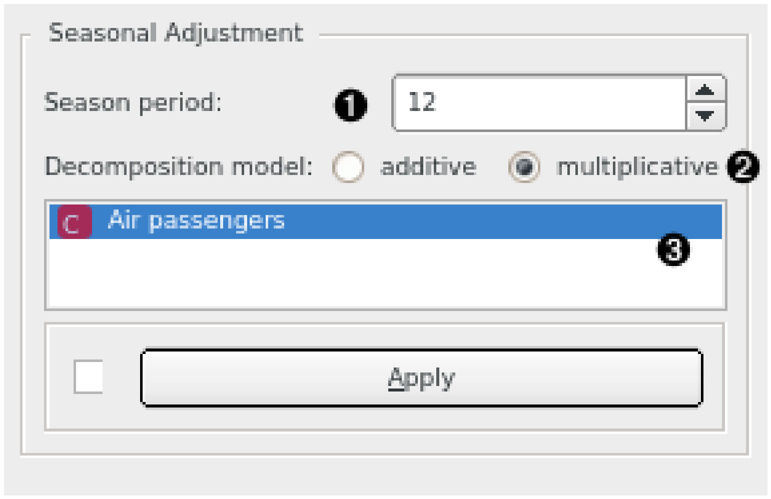
Decompose the time series into seasonal, trend, and residual components.

Inputs

- Time series: Time series as output by As Timeseries widget.

Outputs

- Time series: Original time series with some additional columns: seasonal component, trend component, residual component, and seasonally adjusted time series.



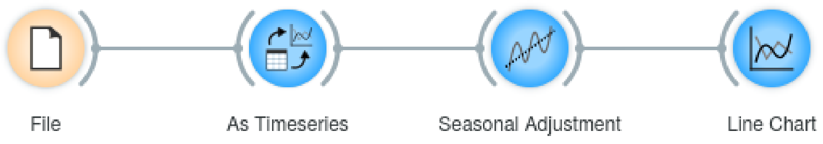
Season period: 12

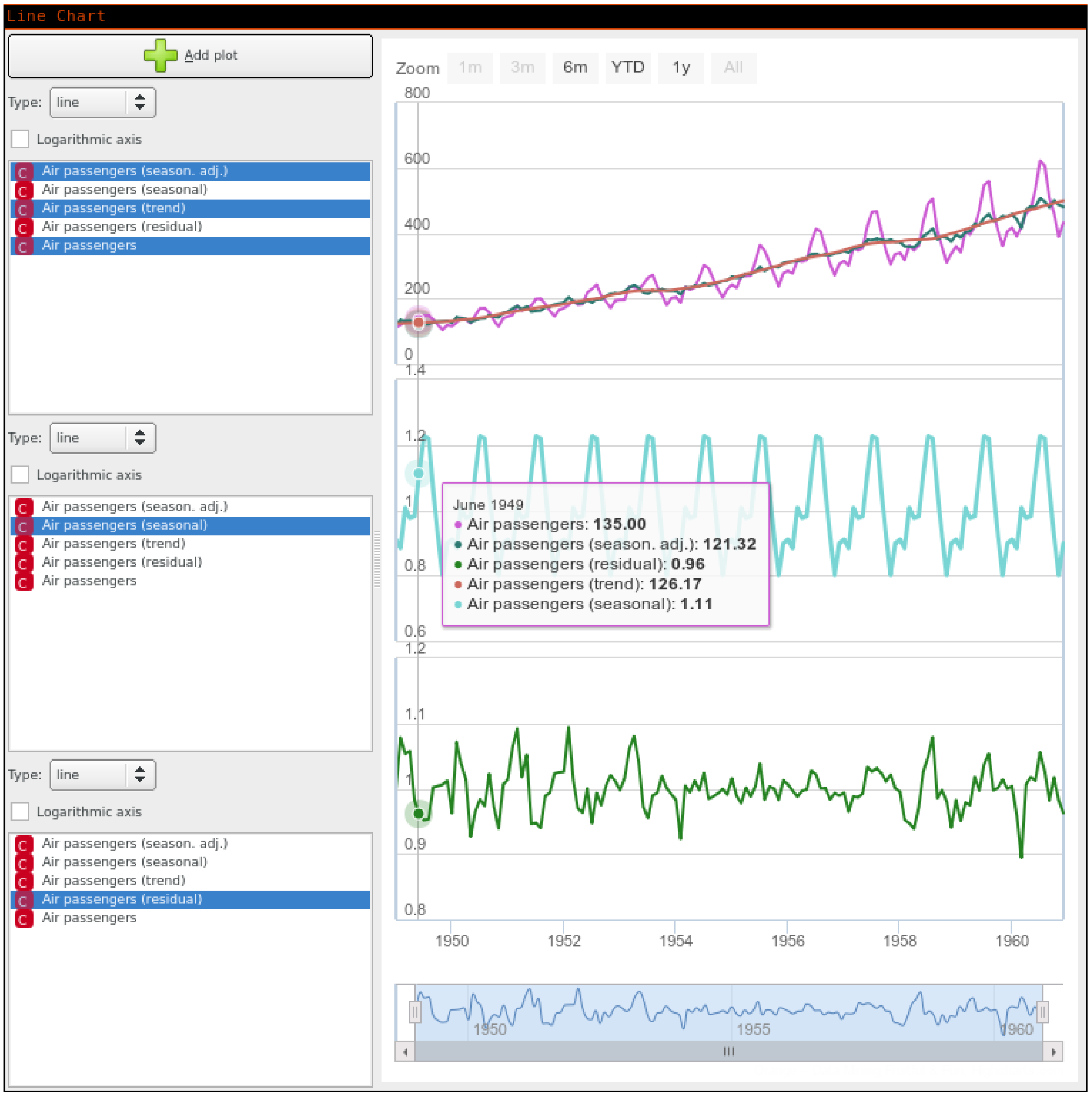
Decomposition model: additive [✓ multiplicative]

Air passengers

1. Length of the season in periods (e.g. 12 for monthly data).
2. Time series decomposition model, additive or multiplicative.
3. The series to seasonally adjust.

1.16.1 Example





June 1949
● Air passengers: **135.00**
● Air passengers (season. adj.): **121.32**
● Air passengers (residual): **0.96**
● Air passengers (trend): **126.17**
● Air passengers (seasonal): **1.11**

See also

Moving Transform

CHAPTER 2

Python Scripting

2.1 Scripting Tutorial

Start by importing the relevant objects:

```
>>> from orangecontrib.timeseries import *
```

Let’s load new `Timeseries`, for example:

```
>>> data = Timeseries.from_file('airpassengers')
>>> np.set_printoptions(precision=1)
```

`Timeseries` object is just an `Orange.data.Table` object with some extensions.

Find more info and function docstrings in the *reference*.

2.1.1 Periodicity

You can compute periodogram values using `periodogram()` or `periodogram_nonequispaced()` (Lomb-Scargle) for non-uniformly spaced time series.

With our air passengers example, calculate the periodogram on the only data-bearing column, which also happens to be a class variable:

```
>>> periods, pgram_values = periodogram(data.Y, detrend='diff')
>>> periods
array([  2.4,    3. ,    4. ,    6. ,   11.9])
>>> pgram_values
array([0.1, 0.2, 0.2, 1. , 0.9])
```

Obviously, 6 and 12 are important periods for this data set.

2.1.2 Autocorrelation

Compute autocorrelation or partial autocorrelation coefficients using `autocorrelation()` or `partial_autocorrelation()` functions. For example:

```
>>> acf = autocorrelation(data.Y)
>>> acf[:4]
array([[12. ,  0.8],
       [24. ,  0.6],
       [36. ,  0.4],
       [48. ,  0.2]])
>>> pacf = partial_autocorrelation(data.Y)
>>> pacf[:4]
array([[ 9. ,  0.2],
       [13. , -0.5],
       [25. , -0.2],
       [40. , -0.1]])
```

2.1.3 Interpolation

Let's say your data is missing some values:

```
>>> data.Y[7:11]
array([148., 136., 119., 104.])
>>> data.Y[7:11] = np.nan
```

You can interpolate those values with one of supported interpolation methods using `interpolate_timeseries()` function:

```
>>> interpolated = interpolate_timeseries(data, method='cubic')
>>> interpolated[7:11].Y
array([151.2, 146.8, 137.8, 127.2])
>>> data = interpolated
```

2.1.4 Seasonal decomposition

To decompose the time series into trend, seasonal and residual components, use `seasonal_decompose()` function:

```
>>> from Orange.data import Domain
>>> passengers = Timeseries.from_table(Domain(['Air passengers'], source=data.domain),
↪ data)
>>> decomposed = seasonal_decompose(passengers, model='multiplicative', period=12)
>>> decomposed.domain
[Air passengers (season. adj.), Air passengers (seasonal), Air passengers (trend),␣
↪Air passengers (residual)]
```

To use this decomposed time series effectively, we just have to add back the time variable that was stripped in the first step above:

```
>>> ts = Timeseries.concatenate((data, decomposed))
>>> ts.time_variable = data.time_variable
```

Just kidding. Use `statsmodels.seasonal.seasonal_decompose()` instead.

2.1.5 Moving transform

It's easy enough to apply moving windows transforms over any raw data in Python. In Orange3-Timeseries, you can use `moving_transform()` function. It accepts a time series object and a transform specification (list of tuples `(Variable, window length, aggregation function)`). For example:

```
>>> spec = [(data.domain['Air passengers'], 10, np.nanmean), ]  # Just 10-year SMA
>>> transformed = moving_transform(data, spec)
>>> transformed.domain
[Month, Air passengers (10; nanmean) | Air passengers]
>>> transformed
[[1949-01-01, 112.000 | 112],
 [1949-02-01, 115.000 | 118],
 [1949-03-01, 120.667 | 132],
 [1949-04-01, 122.750 | 129],
 [1949-05-01, 122.400 | 121],
 ...
]
```

There are a couple of nan-safe aggregation functions available in `orangecontrib.timeseries.agg_funcs` module.

2.1.6 Time series modelling and forecast

There are, as of yet, two models available: ARIMA and VAR. Both models have a common interface, so the usage of one is similar to the other. Let's look at an example. The data we model must have defined a class variable:

```
>>> data = Timeseries.from_file('airpassengers')
>>> data.domain
[Month | Air passengers]
>>> data.domain.class_var
ContinuousVariable(name='Air passengers', number_of_decimals=0)
```

We define the model with its parameters (see the reference for what arguments each model accepts):

```
>>> model = ARIMA((2, 1, 1))
```

Now we fit the data:

```
>>> model.fit(data)
<...ARIMA object at 0x...>
```

After fitting, we can get the forecast along with desired confidence intervals:

```
>>> forecast, ci95_low, ci95_high = model.predict(steps=10, alpha=.05)
```

We can also output the prediction as a `Timeseries` object:

```
>>> forecast = model.predict(10, as_table=True)
>>> forecast.domain
[Air passengers (forecast), Air passengers (95%CI low), Air passengers (95%CI high)]
>>> forecast.X
array([[470.5, 417.8, 523.1],
       [492.6, 414.1, 571.1],
       [498.5, 411.5, 585.4],
       ...
```

(continues on next page)

(continued from previous page)

```
       [492.7, 403. , 582.4],
       [497.1, 407.3, 586.8]])
```

We can examine model's fitted values and residuals with appropriately-named methods:

```
>>> model.fittedvalues(as_table=False)
array([114.7, 121.7, ..., 440.4, 386.8])
>>> model.residuals(as_table=False)
array([ 3.3,  10.3, ..., -50.4,  45.2])
```

We can evaluate the model on in-sample, fitted values:

```
>>> for measure, error in sorted(model.errors().items()):
...     print('{:7s} {:>6.2f}'.format(measure.upper(), error))
MAE      19.66
MAPE      0.08
POCID    58.45
R2        0.95
RMSE     27.06
```

Finally, one should more robustly evaluate their models using cross validation. An example, edited for some clarity:

```
>>> models = [ARIMA((1, 1, 0)), ARIMA((2, 1, 2)), VAR(1), VAR(3)]
>>> model_evaluation(data, models, n_folds=10, forecast_steps=3)  # doctest: +SKIP
[['Model',                    'RMSE', 'MAE', 'MAPE', 'POCID', 'R²', 'AIC', 'BIC'],
 ['ARIMA(1,1,0)',             47.318, 36.803, 0.093, 68.965, 0.625, 1059.3, 1067.4],
 ['ARIMA(1,1,0) (in-sample)', 32.040, 20.340, 0.089, 58.450, 0.927, 1403.4, 1412.3],
 ['ARIMA(2,1,2)',             44.659, 28.332, 0.075, 72.413, 0.666, 1032.8, 1049.2],
 ['ARIMA(2,1,2) (in-sample)', 25.057, 16.159, 0.070, 59.859, 0.955, 1344.0, 1361.8],
 ['VAR(1)',                   63.185, 45.553, 0.118, 68.965, 0.332, 28.704, 28.849],
 ['VAR(1) (in-sample)',       31.316, 19.001, 0.084, 54.929, 0.930, 29.131, 29.255],
 ['VAR(3)',                   46.210, 28.526, 0.085, 82.758, 0.643, 28.140, 28.482],
 ['VAR(3) (in-sample)',       25.642, 18.010, 0.072, 61.428, 0.953, 28.406, 28.698]]
```

2.1.7 Granger Causality

Use `granger_causality()` to estimate causality between series. A synthetic example:

```
>>> series = np.arange(100)
>>> X = np.column_stack((series, np.roll(series, 1), np.roll(series, 3)))
>>> threecol = Timeseries.from_numpy(Domain.from_numpy(X), X)
>>> for lag, ante, cons in granger_causality(threecol, 10):
...     if lag > 1:
...         print('Series {cons} lags by {ante} by {lag} lags.'.format(**locals()))
...
Series Feature 1 lags by Feature 2 by 3 lags.
Series Feature 2 lags by Feature 3 by 4 lags.
```

Use this knowledge wisely.

2.2 Module Reference

`functions.`**`r2`**`(`*true*, *pred*`)`

Coefficient of determination (R^2)

`functions.`**`rmse`**(*true*, *pred*)
Root mean squared error

`functions.`**`mape`**(*true*, *pred*)
Mean absolute percentage error

`functions.`**`mae`**(*true*, *pred*)
Median absolute error

`functions.`**`pocid`**(*true*, *pred*)
Prediction on change of direction

`functions.`**`periodogram`**(*x*, **args*, *detrend='diff'*, ***kwargs*)
Return periodogram of signal *x*.

Parameters

- **x** (*array_like*) – A 1D signal.
- **detrend** (*'diff' or False or int*) – Remove trend from x. If int, fit and subtract a polynomial of this order. See also: *statsmodels.tsa.detrend*.
- **kwargs** (*args,*) – As accepted by *scipy.signal.periodogram*.

Returns

- **periods** (*array_like*) – The periods at which the spectral density is calculated.
- **pgram** (*array_like*) – Power spectral density of x.

`functions.`**`periodogram_nonequispaced`**(*times*, *x*, ***, *freqs=None*, *period_low=None*, *period_high=None*, *n_periods=1000*, *detrend='linear'*)
Compute the Lomb-Scargle periodogram for non-equispaced timeseries.

Parameters

- **times** (*array_like*) – Sample times.
- **x** (*array_like*) – A 1D signal.
- **freqs** (*array_like, optional*) – **Angular** frequencies for output periodogram.
- **period_low** (*float*) – If *freqs* not provided, the lowest period for which to look for periodicity. Defaults to 5th percentile of time difference between observations.
- **period_high** (*float*) – If *freqs* not provided, the highest period for which to look for periodicity. Defaults to 80th percentile of time difference of observations, or 200*period_low, whichever is larger.
- **n_periods** (*int*) – Number of periods between period_low and period_high to try.
- **detrend** (*'diff' or False or int*) – Remove trend from x. If int, fit and subtract a polynomial of this order. See also: *statsmodels.tsa.detrend*.

Returns

- **periods** (*array_like*) – The periods at which the spectral density is calculated.
- **pgram** (*array_like*) – Lomb-Scargle periodogram.

Notes

Read also: https://jakevdp.github.io/blog/2015/06/13/lomb-scargle-in-python/#lomb-scargle-algorithms-in-python

`functions.`**`autocorrelation`**(*x*, **args*, *unbiased=True*, *nlags=None*, *fft=True*, ***kwargs*)
Return autocorrelation function of signal *x*.

Parameters

- **`x`** (*`array_like`*) – A 1D signal.
- **`nlags`** (*`int`*) – The number of lags to calculate the correlation for (default .9*len(x))
- **`fft`** (*`bool`*) – Compute the ACF via FFT.
- **`kwargs`** (*`args,`*) – As accepted by *statsmodels.tsa.stattools.acf*.

Returns

- **acf** (*array*) – Autocorrelation function.
- **confint** (*array, optional*) – Confidence intervals if alpha kwarg provided.

`functions.`**`partial_autocorrelation`**(*x*, **args*, *nlags=None*, *method='ldb'*, ***kwargs*)
Return partial autocorrelation function (PACF) of signal *x*.

Parameters

- **`x`** (*`array_like`*) – A 1D signal.
- **`nlags`** (*`int`*) – The number of lags to calculate the correlation for (default: min(len(x)//2 - 1, len(x) - 1))
- **`kwargs`** (*`args,`*) – As accepted by *statsmodels.tsa.stattools.pacf*.

Returns

- **acf** (*array*) – Partioal autocorrelation function.
- **confint** (*optional*) – As returned by *statsmodels.tsa.stattools.pacf*.

`functions.`**`interpolate_timeseries`**(*data*, *method='linear'*, *multivariate=False*)
Return a new Timeseries (Table) with nan values interpolated.

Parameters

- **`data`** (*`Orange.data.Table`*) – A table to interpolate.
- **`method`** (*`str {'linear', 'cubic', 'nearest', 'mean'}`*) – The interpolation method to use.
- **`multivariate`** (*`bool`*) – Whether to perform multivariate (2d) interpolation first. Univariate interpolation of same method is always performed as a final step.

Returns **series** – A table with nans in original replaced with interpolated values.

Return type Timeseries

`functions.`**`seasonal_decompose`**(*data*, *model='multiplicative'*, *period=12*, ***, *callback=None*)
Return table of decomposition components of original features and original features seasonally adjusted.

Parameters

- **`data`** (*`Timeseries`*) – A table of featres to decompose/adjust.
- **`model`** (*`str {'additive', 'multiplicative'}`*) – A decompostition model. See: https://en.wikipedia.org/wiki/Decomposition_of_time_series
- **`period`** (*`int`*) – The period length of season.
- **`callback`** (*`callable`*) – Optional callback to call (with no parameters) after each iteration.

Returns **table** – Table with columns: original series seasonally adjusted, original series' seasonal components, trend components, and residual components.

Return type Timeseries

`functions.`**`granger_causality`**`(`*data*, *max_lag=10*, *alpha=0.05*, ***, *callback=None*`)`

Return results of Granger-causality tests.

Parameters

- **`data`** (*`Timeseries`*) – A table of features to compute Granger causality between.
- **`max_lag`** (*`int`*) – The maximum lag to compute Granger-causality for.
- **`alpha`** (*`float in (0, 1)`*) – Confidence of test is 1 - alpha.
- **`callback`** (*`callable`*) – A callback to call in each iteration with ratio of completion.

Returns **res** – Each internal list is [lag, p-value, antecedent, consequent] where lag is the minimum lag at which antecedent feature in data is Granger-causal for the consequent feature in data.

Return type list of lists

`functions.`**`moving_transform`**`(`*data*, *spec*, *fixed_wlen=0*`)`

Return data transformed according to spec.

Parameters

- **`data`** (*`Timeseries`*) – A table with features to transform.
- **`spec`** (*`list of lists`*) – A list of lists [feature:Variable, window_length:int, function:callable].
- **`fixed_wlen`** (*`int`*) – If not 0, then window_length in spec is disregarded and this length is used. Also the windows don't shift by one but instead align themselves side by side.

Returns **transformed** – A table of original data its transformations.

Return type Timeseries

`functions.`**`model_evaluation`**`(`*data*, *models*, *n_folds*, *forecast_steps*, ***, *callback=None*`)`

Evaluate models on data.

Parameters

- **`data`** (*`Timeseries`*) – The timeseries to model. Must have a class variable that is used for prediction and scoring.
- **`models`** (*`list`*) – List of models (objects with fit() and predict() methods) to try.
- **`n_folds`** (*`int`*) – Number of iterations.
- **`forecast_steps`** (*`int`*) – Number of forecast steps at each iteraction.
- **`callback`** (*`callable, optional`*) – Optional argument-less callback to call after each iteration.

Returns **results** – A table with horizontal and vertical headers and results. Print it to see it.

Return type list of lists

`class` `agg_funcs.`**`Sum`**

`class` `agg_funcs.`**`Product`**

`class` `agg_funcs.`**`Mean`**

`class` `agg_funcs.`**`Count_nonzero`**

class agg_funcs.**Count_defined**

class agg_funcs.**Max**

class agg_funcs.**Min**

class agg_funcs.**Median**

class agg_funcs.**Std_deviation**

class agg_funcs.**Variance**

class agg_funcs.**Mode**

class agg_funcs.**Cumulative_sum**

class agg_funcs.**Cumulative_product**

class agg_funcs.**Harmonic_mean**

class agg_funcs.**Geometric_mean**

class agg_funcs.**Weighted_MA**

class agg_funcs.**Exponential_MA**

class agg_funcs.**Concatenate**

class agg_funcs.**AggDesc**(*transform*, *disc*, *time*)

disc
Alias for field number 1

time
Alias for field number 2

transform
Alias for field number 0

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index