Khiva Documentation

Release v0.3.0

Shapelets.io

Jun 11, 2019

Contents:

CHAPTER 1

Khiva API

This is the list of namespaces that comprise the Khiva library.

1.1 Namespace Array

class KhivaArray : **public** IDisposable
Khiva *KhivaArray* Class.

Public Types

enum DType
KHIVA array available types.

Values:

F32
Floating point of single precision. Khiva.DType.

C32
Complex floating point of single precision. Khiva.DType.

F64
Floating point of double precision. Khiva.DType.

C64
Complex floating point of double precision. Khiva.DType.

B8
Boolean. Khiva.DType.

S32
32 bits Int. Khiva.DType.

U32
32 bits Unsigned Int. Khiva.DType.

U8
8 bits Unsigned Int. Khiva.DType.

S64
64 bits Integer. Khiva.DType.

U64
64 bits Unsigned Int. Khiva.DType.

S16
16 bits Int. Khiva.DType.

U16
16 bits Unsigned Int. Khiva.DType.

Public Functions

void Khiva.KhivaArray.Dispose()
Dispose the *KhivaArray*.

T [] Khiva.KhivaArray.GetData1D< T >()
Get the data of a 4 dimensional khiva array.

Return 1 dimensional array containing the data of the khiva array.

Template Parameters

- T: The type of the elements of the array.

T [,] Khiva.KhivaArray.GetData2D< T >()
Get the data of a 2 dimensional khiva array.

Return 2 dimensional array containing the data of the khiva array.

Template Parameters

- T: The type of the elements of the array.

T [,,] Khiva.KhivaArray.GetData3D< T >()
Get the data of a 3 dimensional khiva array.

Return 3 dimensional array containing the data of the khiva array.

Template Parameters

- T: The type of the elements of the array.

T [,,,] Khiva.KhivaArray.GetData4D< T >()
Get the data of a 4 dimensional khiva array.

Return 4 dimensional array containing the data of the khiva array.

Template Parameters

- T: The type of the elements of the array.

void Khiva.KhivaArray.Display()
Displays an *KhivaArray*.

override bool Khiva.KhivaArray.Equals(object o)
Equals object method

Return

Parameters

- `o`: Object to compare

override int Khiva.KhivaArray.GetHashCode()
GetHashCode object method

Return Hashcode of the *KhivaArray*

KhivaArray Khiva.KhivaArray.Transpose(bool conjugate = false)
Transposes array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `conjugate`: If true a conjugate transposition is performed.

KhivaArray Khiva.KhivaArray.Col(int index)
Retrieves a given column of array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `index`: The column to be retrieved.

KhivaArray Khiva.KhivaArray.Cols(int first, int last)
Retrieves a subset of columns of array, starting at first and finishing at last, both inclusive.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `first`: Start of the subset of columns to be retrieved.
- `last`: End of the subset of columns to be retrieved.

KhivaArray Khiva.KhivaArray.Row(int index)
Retrieves a given row of array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `index`: The row to be retrieved.

KhivaArray Khiva.KhivaArray.Rows(int first, int last)
Retrieves a subset of rows of array, starting at first and finishing at last, both inclusive.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `first`: Start of the subset of rows to be retrieved.
- `last`: End of the subset of rows to be retrieved.

KhivaArray Khiva.KhivaArray.MatMul(KhivaArray rhs)
Performs a matrix multiplication of lhs and rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `rhs`: Right-hand side KHIVA array for the operation.

KhivaArray Khiva.KhivaArray.Copy()
Performs a deep copy of array.

Return KHIVA *KhivaArray* which contains a copy of array.

KhivaArray Khiva.KhivaArray.As(int type)
Changes the type of array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `type`: Target type of the output array.

Property

property Khiva::KhivaArray::Reference
Getters and setters of the Reference parameter.

property Khiva::KhivaArray::Dims
Gets the *KhivaArray* dimensions.

property Khiva::KhivaArray::ArrayType
Gets the type of the array.

Public Static Functions

static KhivaArray Khiva.KhivaArray.CreateZeros< T >(long [] dims, uint nDims, bool dou
Creates *KhivaArray* of zeros.

Return *KhivaArray* created.

Template Parameters

- `T`: Type of the elements of the array.

Parameters

- `dims`: Cardinality of dimensions of the data.
- `nDims`: Number of dimensions of the data.
- `doublePrecision`: If Complex array has double precision. Default to false.

static KhivaArray Khiva.KhivaArray.Create(IntPtr reference)
Creates a khiva array object from reference.

Return *KhivaArray* created.

Parameters

- `reference`: Reference from which create the array.

static KhivaArray Khiva.KhivaArray.Create(KhivaArray other)
Creates a khiva array object from copy.

Return *KhivaArray* created.

Parameters

- `other`: *KhivaArray* to copy.

static unsafe KhivaArray Khiva.KhivaArray.Create< T >(T [] values, bool doublePrecisio
Creates a khiva array object.

Return *KhivaArray* created.

Template Parameters

- `T`: Type of the elements of the array.

Parameters

- `values`: 1 dimensional array with the data.
- `doublePrecision`: If Complex array has double precision. Default to false.

static unsafe KhivaArray Khiva.KhivaArray.Create< T >(T values[,], bool doublePrecisio
Creates a khiva array object.

Return *KhivaArray* created.

Template Parameters

- `T`: Type of the elements of the array.

Parameters

- `values`: 2 dimensional array with the data.
- `doublePrecision`: If Complex array has double precision. Default to false.

static unsafe KhivaArray Khiva.KhivaArray.Create< T >(T values[,,], bool doublePrecisi
Creates a khiva array object.

Return *KhivaArray* created

Template Parameters

- `T`: Type of the elements of the array.

Parameters

- `values`: 3 dimensional array with the data.
- `doublePrecision`: If Complex array has double precision. Default to false.

static unsafe KhivaArray Khiva.KhivaArray.Create< T >(T values[,,,], bool doublePrecis
Creates a khiva array object.

Return *KhivaArray* created.

Template Parameters

- `T`: Type of the elements of the array.

Parameters

- `values`: 4 dimensional array with the data.
- `doublePrecision`: If Complex array has double precision. Default to false.

static KhivaArray Khiva.KhivaArray.operator+(KhivaArray lhs, KhivaArray rhs)
Adds two arrays.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator*(KhivaArray lhs, KhivaArray rhs)
Multiplies two arrays.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator-(KhivaArray lhs, KhivaArray rhs)
Subtracts two arrays.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator/(KhivaArray lhs, KhivaArray rhs)
Divides two arrays.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator%(KhivaArray lhs, KhivaArray rhs)
Performs the modulo operation of lhs by rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.Pow(KhivaArray lhs, KhivaArray rhs)
Powers lhs with rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator &(KhivaArray lhs, KhivaArray rhs)
Performs an AND operation (element-wise) with lhs and rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator|(KhivaArray lhs, KhivaArray rhs)
Performs an OR operation (element-wise) with lhs and rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator^(KhivaArray lhs, KhivaArray rhs)
Performs an eXclusive-OR operation (element-wise) with lhs and rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator<<(KhivaArray lhs, int shift)
Performs a left bit shift operation (element-wise) to array as many times as specified in the parameter n.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: array KHIVA *KhivaArray* to shift.
- `shift`: Number of bits to be shifted.

static KhivaArray Khiva.KhivaArray.operator>>(KhivaArray lhs, int shift)
Performs a right bit shift operation (element-wise) to array as many times as specified in the parameter n.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: array KHIVA *KhivaArray* to shift.
- `shift`: Number of bits to be shifted.

static KhivaArray Khiva.KhivaArray.operator-(KhivaArray rhs)
Unary minus of one array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator!(KhivaArray lhs)
Logical NOT operation to array.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: KHIVA *KhivaArray* to negate.

static KhivaArray Khiva.KhivaArray.operator<(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if lhs is lower than rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator>(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if lhs is greater than rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator<=(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if lhs is lower or equal than rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator>=(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if lhs is greater or equal than rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator==(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if rhs is equal to rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

static KhivaArray Khiva.KhivaArray.operator!=(KhivaArray lhs, KhivaArray rhs)
Compares (element-wise) if lhs is not equal to rhs.

Return KHIVA *KhivaArray* with the result of this operation.

Parameters

- `lhs`: Left-hand side KHIVA array for the operation.
- `rhs`: Right-hand side KHIVA array for the operation.

Private Functions

void Khiva.KhivaArray.CleanUp()
Clean up the array.

Khiva.KhivaArray.~KhivaArray()
Destroy *KhivaArray*.

Khiva.KhivaArray.KhivaArray()
Creates empty *KhivaArray*.

void Khiva.KhivaArray.DeleteArray()
Decreases the references count of the given array.

1.2 Namespace Clustering

class Clustering
Khiva *Clustering* class containing several clustering methods.

Public Static Functions

static Tuple<KhivaArray, KhivaArray> Khiva.Clustering.KMeans(KhivaArray arr, int k, fl
Calculates the k-means algorithm.

[1] S.Lloyd. 1982. Least squares quantization in PCM.IEEE Transactions on Information Theory, 28, 2, Pages 129-137.

Return Tuple with the resulting means or centroids and the resulting labels of each time series which is the closest centroid.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `k`: The number of means to be computed.
- `tolerance`: The error tolerance to stop the computation of the centroids.
- `maxIterations`: The maximum number of iterations allowed.

static Tuple<KhivaArray, KhivaArray> Khiva.Clustering.KShape(KhivaArray arr, int k, fl
Calculates the k-shape algorithm.

[1] John Paparrizos and Luis Gravano. 2016. k-Shape: Efficient and Accurate *Clustering* of Time Series. SIGMOD Rec. 45, 1 (June 2016), 69-76.

Return Tuple with the resulting means or centroids and the resulting labels of each time series which is the closest centroid.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

- `k`: The number of means to be computed.
- `tolerance`: The error tolerance to stop the computation of the centroids.
- `maxIterations`: The maximum number of iterations allowed.

1.3 Namespace Dimensionality

class Dimensionality

Khiva *Dimensionality* class containing several dimensionality reduction methods.

Public Static Functions

static KhivaArray Khiva.Dimensionality.Paa(KhivaArray arr, int bins)

Piecewise Aggregate Approximation (PAA) approximates a time series X of length n into vector $\bar{X} = (\bar{x}_1, \ldots, \bar{x}_M)$ of any arbitrary length $M \leq n$ where each of $\bar{x}_i$ is calculated as follows:

$$\bar{x}_i = \frac{M}{n} \sum_{j=n/M(i-1)+1}^{(n/M)i} x_j.$$

Which simply means that in order to reduce the dimensionality from n to M, we first divide the original time series into M equally sized frames and secondly compute the mean values for each frame. The sequence assembled from the mean values is the PAA approximation (i.e., transform) of the original time series.

Return An array of points with the reduced dimensionality.

Parameters

- `arr`: Set of points.
- `bins`: Sets the total number of divisions.

static KhivaArray Khiva.Dimensionality.Pip(KhivaArray arr, int numberIps)

Calculates the number of Perceptually Important Points (PIP) in the time series.

[1] Fu TC, Chung FL, Luk R, and Ng CM. Representing financial time series based on data point importance. Engineering Applications of Artificial Intelligence, 21(2):277-300, 2008.

Return *KhivaArray* with the most Perceptually Important number_ips.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series.
- `numberIps`: The number of points to be returned.

static KhivaArray Khiva.Dimensionality.PlaBottomUp(KhivaArray arr, float maxError)

Applies the Piecewise Linear Approximation (PLA BottomUP) to the time series.

[1] Zhu Y, Wu D, Li Sh (2007). A Piecewise Linear Representation Method of Time Series Based on Feature Points. Knowledge-Based Intelligent Information and Engineering Systems 4693:1066-1072.

Return The reduced number of points.

Parameters

- `arr`: Expects a khiva_array containing the set of points to be reduced. The first component of the points in the first column and the second component of the points in the second column.
- `maxError`: The maximum approximation error allowed.

static KhivaArray Khiva.Dimensionality.PlaSlidingWindow(KhivaArray arr, float maxError

Applies the Piecewise Linear Approximation (PLA Sliding Window) to the time series.

[1] Zhu Y, Wu D, Li Sh (2007). A Piecewise Linear Representation Method of Time Series Based on Feature Points. Knowledge-Based Intelligent Information and Engineering Systems 4693:1066-1072.

Return The reduced number of points.

Parameters

- `arr`: Expects a khiva_array containing the set of points to be reduced. The first component of the points in the first column and the second component of the points in the second column.
- `maxError`: The maximum approximation error allowed.

static KhivaArray Khiva.Dimensionality.RamerDouglasPeucker(KhivaArray points, double e

The Ramer–Douglas–Peucker algorithm (RDP) is an algorithm for reducing the number of points in a curve that is approximated by a series of points. It reduces a set of points depending on the perpendicular distance of the points and epsilon, the greater epsilon, more points are deleted.

[1] Urs Ramer, "An iterative procedure for the polygonal approximation of plane curves", Computer Graphics and Image Processing, 1(3), 244–256 (1972) doi:10.1016/S0146-664X(72)80017-0.

[2] David Douglas &; Thomas Peucker, "Algorithms for the reduction of the number of points required to represent a

digitized line or its caricature", The Canadian Cartographer 10(2), 112–122 (1973). doi:10.3138/FM57-6770-U75U-7727

Return *KhivaArray* with the x-coordinates and y-coordinates of the selected points (x in column 0 and y in column 1).

Parameters

- `points`: *KhivaArray* with the x-coordinates and y-coordinates of the input points (x in column 0 and y in column 1).
- `epsilon`: It acts as the threshold value to decide which points should be considered meaningful or not.

static KhivaArray Khiva.Dimensionality.SAX(KhivaArray arr, int alphabetSize)

Symbolic Aggregate approXimation (SAX). It transforms a numeric time series into a time series of symbols with the same size. The algorithm was proposed by Lin et al.) and extends the PAA-based approach inheriting the original algorithm simplicity and low computational complexity while providing satisfactory sensitivity and selectivity in range query processing. Moreover, the use of a symbolic representation opened a door to the existing wealth of data-structures and string-manipulation algorithms in computer science such as hashing, regular expression, pattern matching, suffix trees, and grammatical inference.

[1] Lin, J., Keogh, E., Lonardi, S. &; Chiu, B. (2003) A Symbolic Representation of Time Series, with Implications for Streaming Algorithms. In proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. San Diego, CA. June 13.

Return An array of symbols.

Parameters

- `arr`: *KhivaArray* with the input time series.

- `alphabetSize`: Number of element within the alphabet.

static KhivaArray Khiva.Dimensionality.Visvalingam(KhivaArray points, int numPoints)
Reduces a set of points by applying the Visvalingam method (minimum triangle area) until the number of points is reduced to numPoints.

[1] M. Visvalingam and J. D. Whyatt, Line generalisation by repeated elimination of points, The Cartographic Journal, 1993.

Return *KhivaArray* with the x-coordinates and y-coordinates of the selected points (x in column 0 and y in column 1).

Parameters

- `points`: *KhivaArray* with the x-coordinates and y-coordinates of the input points (x in column 0 and y in column 1).
- `numPoints`: Sets the number of points returned after the execution of the method.

1.4 Namespace Distances

class Distances
Khiva *Distances* class containing distances methods.

Public Static Functions

static KhivaArray Khiva.Distances.Dtw(KhivaArray arr)
Calculates the Dynamic Time Warping Distance.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Distances.Euclidean(KhivaArray arr)
Calculates euclidean distances between time series.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Distances.Hamming(KhivaArray arr)
Calculates Hamming distances between time series.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Distances.Manhattan(KhivaArray arr)
Calculates Manhattan distances between time series.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Distances.Sbd(KhivaArray arr)
Calculates the Shape-Based distance (SBD). It computes the normalized cross-correlation and it returns 1.0 minus the value that maximizes the correlation value between each pair of time series.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Distances.SquaredEuclidean(KhivaArray arr)
Calculates the Shape-Based distance (SBD). It computes the normalized cross-correlation and it returns 1.0 minus the value that maximizes the correlation value between each pair of time series.

Return An upper triangular matrix where each position corresponds to the distance between two time series. Diagonal elements will be zero. For example: Position row 0 column 1 records the distance between time series 0 and time series 1.

Parameters

- `arr`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

1.5 Namespace Features

class Features
Khiva *Features* class containing a number of features that can be extracted from time series. All the methods operate with instances of the ARRAY class as input and output.

Public Static Functions

static KhivaArray Khiva.Features.AbsEnergy(KhivaArray array)
Calculates the sum over the square values of the time series.

Return An array with the same dimensions as array, whose values (time series in dimension 0) contains the sum of the squares values in the time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.AbsoluteSumOfChanges(KhivaArray array)

Calculates the sum over the absolute value of consecutive changes in the time series.

Return An array with the same dimensions as array, whose values (time series in dimension 0) contains absolute value of consecutive changes in the time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.AggregatedAutoCorrelation(KhivaArray array, int aggre

Calculates the value of an aggregation function f_agg (e.g. var or mean) of the autocorrelation (Compare to http://en.wikipedia.org/wiki/Autocorrelation#Estimation), taken over different all possible lags(1 to length of x).

Return An array whose values contains the aggregated correlation for each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `aggregationFunction`: Function to be used in the aggregation.It receives an integer which indicates the function to be applied: { 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, 5 : var, default : mean }

static Tuple<KhivaArray, KhivaArray, KhivaArray, KhivaArray, KhivaArray> Khiva.Feature

Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.

Parameters

- `array`: The time series to calculate the features of
- `chunkSize`: The chunk size used to aggregate the data.
- `aggregationFunction`: Function to be used in the aggregation. It receives an integer which indicates the function to be applied: { 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, default : mean } /param>

Return Tuple with the slope of the regression line, the intercept of the regression line, the correlation coefficient, the two-sided p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic and the standard error of the estimated gradient.

static KhivaArray Khiva.Features.ApproximateEntropy(KhivaArray array, int m, float r)

Calculates a vectorized Approximate entropy algorithm. https://en.wikipedia.org/wiki/Approximate_entropy For short time-series this method is highly dependent on the parameters, but should be stable for N > 2000, see: Yentes et al. (2012) - The Appropriate Use of Approximate Entropy and Sample Entropy with Short Data Sets Other shortcomings and alternatives discussed in: Richman &; Moorman (2000) - Physiological time-series analysis using approximate entropy and sample entropy.

Return The vectorized approximate entropy for all the input time series in array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `m`: Length of compared run of data.
- `r`: Filtering level, must be positive.

static KhivaArray Khiva.Features.CrossCovariance(KhivaArray xss, KhivaArray yss, bool
Calculates the cross-covariance of the given time series.

Return The cross-covariance value for the given time series.

Parameters

- `xss`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `yss`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `unbiased`: Determines whether it divides by n - lag (if true) or n(if false).

static KhivaArray Khiva.Features.AutoCovariance(KhivaArray array, bool unbiased = fals
Calculates the auto-covariance the given time series.

Return The auto-covariance value for the given time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `unbiased`: Determines whether it divides by n - lag (if true) or n(if false).

static KhivaArray Khiva.Features.CrossCorrelation(KhivaArray xss, KhivaArray yss, bool
Calculates the cross-correlation of the given time series.

Return The cross-correlation value for the given time series.

Parameters

- `xss`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `yss`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `unbiased`

static KhivaArray Khiva.Features.AutoCorrelation(KhivaArray array, long maxLag, bool u
Calculates the autocorrelation of the specified lag for the given time series.

Return The autocorrelation value for the given time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `maxLag`: The maximum lag to compute.
- `unbiased`: Determines whether it divides by n - lag (if true) or n (if false)

static KhivaArray Khiva.Features.BinnedEntropy(KhivaArray array, int maxBins)
Calculates the binned entropy for the given time series and number of bins.

Return The binned entropy value for the given time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `maxBins`: The number of bins.

static KhivaArray Khiva.Features.C3(KhivaArray array, long lag)
Calculates the Schreiber, T. and Schmitz, A. (1997) measure of non-linearity for the given time series.

Return The non-linearity value for the given time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `lag`: The lag.

static KhivaArray Khiva.Features.CidCe(KhivaArray array, bool zNormalize)
Calculates an estimate for the time series complexity defined by Batista, Gustavo EAPA, et al(2014). (A more complex time series has more peaks, valleys, etc.).

Return The complexity value for the given time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `zNormalize`: Controls whether the time series should be z-normalized or not.

static KhivaArray Khiva.Features.CountAboveMean(KhivaArray array)
Calculates the number of values in the time series that are higher than the mean.

Return The number of values in the time series that are higher than the mean.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.CountBelowMean(KhivaArray array)
Calculates the number of values in the time series that are lower than the mean.

Return The number of values in the time series that are lower than the mean.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.CwtCoefficients(KhivaArray array, KhivaArray width, i
Calculates a Continuous wavelet transform for the Ricker wavelet, also known as the "Mexican hat wavelet" which is defined by:

.. math:: {2}{{3a} ^{ {1} { 4 }}} (1 - {x^2}{a^2}) exp(-{ x ^ 2}{2a^2})

where :math:`a` is the width parameter of the wavelet function.

This feature calculator takes three different parameter: widths, coeff and w.The feature calculator takes all the different widths arrays and then calculates the cwt one time for each different width array.Then the

values for the different coefficient for coeff and width w are returned. (For each dic in param one feature is returned).

Return Result of calculated coefficients.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `width`: *KhivaArray* that contains all different widths.
- `coeff`: Coefficient of interest.
- `w`: Width of interest.

static KhivaArray Khiva.Features.EnergyRatioByChunks(KhivaArray array, long numSegment

Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series. segment_focus should be lower than the number of segments.

Return The energy ratio by chunk of the time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `numSegments`: The number of segments to divide the series into.
- `segmentFocus`: The segment number (starting at zero) to return a feature on.

static KhivaArray Khiva.Features.FftAggregated(KhivaArray array)

Calculates the spectral centroid(mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.

Return The spectral centroid (mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static Tuple<KhivaArray, KhivaArray, KhivaArray, KhivaArray> Khiva.Features.FftCoeffic

Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm.

Return Tuple with the real part of the coefficient, the imaginary part of the coefficient, the absolute value of the coefficient and the angle of the coefficient.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `coefficient`: The coefficient to extract from the FFT.

static KhivaArray Khiva.Features.FirstLocationOfMaximum(KhivaArray array)

Calculates the first relative location of the maximal value for each time series.

Return The first relative location of the maximum value to the length of the time series, for each time series.

Parameters

- `array`: array Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.FirstLocationOfMinimum(KhivaArray array)
Calculates the first location of the minimal value of each time series. The position is calculated relatively to the length of the series.

Return The first relative location of the minimal value of each series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.FriedrichCoefficients(KhivaArray array, int m, float
Coefficients of polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model:

$$\dot{(}x)(t) = h(x(t)) + R(N)(0, 1)$$

as described by[1]. For short time series this method is highly dependent on the parameters.

[1] Friedrich et al. (2000): Physics Letters A 271, p. 217-222 Extracting model equations from experimental data.

Return The coefficients for each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `m`: Order of polynom to fit for estimating fixed points of dynamics.
- `r`: Number of quantiles to use for averaging.

static KhivaArray Khiva.Features.HasDuplicates(KhivaArray array)
Calculates if the input time series contain duplicated elements.

Return *KhivaArray* containing True if the time series contains duplicated elements and false otherwise.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.HasDuplicateMax(KhivaArray array)
Calculates if the maximum within input time series is duplicated.

Return *KhivaArray* containing True if the maximum value of the time series is duplicated and false otherwise.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.HasDuplicateMin(KhivaArray array)
Calculates if the minimum of the input time series is duplicated.

Return *KhivaArray* containing True if the minimum of the time series is duplicated and false otherwise.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.IndexMassQuantile(KhivaArray array, float q)
Calculates the index of the max quantile.

Return The index of the max quantile q.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `q`: The quantile.

static KhivaArray Khiva.Features.Kurtosis(KhivaArray array)
Returns the kurtosis of array (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).

Return The kurtosis of each array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.LargeStandardDeviation(KhivaArray array, float r)
Checks if the time series within array have a large standard deviation.

Return *KhivaArray* containing True for those time series in array that have a large standard deviation.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `r`: The threshold.

static KhivaArray Khiva.Features.LastLocationOfMaximum(KhivaArray array)
Calculates the last location of the maximum value of each time series. The position is calculated relatively to the length of the series.

Return The last relative location of the maximum value of each series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.LastLocationOfMinimum(KhivaArray array)
Calculates the last location of the minimum value of each time series. The position is calculated relatively to the length of the series.

Return The last relative location of the minimum value of each series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.Length(KhivaArray array)
Returns the length of the input time series.

Return The length of the time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static Tuple<KhivaArray, KhivaArray, KhivaArray, KhivaArray, KhivaArray> Khiva.Feature
Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.

Return Tuple with the pvalues for all time series, the rvalues for all time series, the intercept values for all time series, the slope for all time series and the stderr values for all time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.LocalMaximals(KhivaArray array)
Calculates all Local Maximals fot the time series in array.

Return The calculated local maximals for each time series in array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.LongestStrikeAboveMean(KhivaArray array)
Calculates the length of the longest consecutive subsequence in array that is bigger than the mean of array.

Return The length of the longest consecutive subsequence in the input time series that is bigger than the mean.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.LongestStrikeBelowMean(KhivaArray array)
Calculates the length of the longest consecutive subsequence in array that is below the mean of array.

Return The length of the longest consecutive subsequence in the input time series that is below the mean.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.MaxLangevinFixedPoint(KhivaArray array, int m, float
Largest fixed point of dynamics $\max_x h(x) = 0$ estimated from polynomial $h(x)$, which has been fitted to the deterministic dynamics of Langevin model

$$\dot(x)(t) = h(x(t)) + R(N)(0, 1)$$

as described by Friedrich et al. (2000): Physics Letters A 271, p. 217-222 *Extracting model equations from experimental data.

Return Largest fixed point of deterministic dynamics.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `m`: Order of polynom to fit for estimating fixed points of dynamics.
- `r`: Number of quantiles to use for averaging.

static KhivaArray Khiva.Features.Maximum(KhivaArray array)
Calculates the maximum value for each time series within array.

Return The maximum value of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.Mean(KhivaArray array)
Calculates the mean value for each time series within array.

Return The mean value of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.MeanAbsoluteChange(KhivaArray array)
Calculates the mean over the absolute differences between subsequent time series values in array.

Return The maximum value of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.MeanChange(KhivaArray array)
Calculates the mean over the differences between subsequent time series values in array.

Return The mean over the differences between subsequent time series values.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.MeanSecondDerivativeCentral(KhivaArray array)
Calculates mean value of a central approximation of the second derivative for each time series in array.

Return The mean value of a central approximation of the second derivative for each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.Median(KhivaArray array)
Calculates the median value for each time series within array.

Return The median value of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.Minimum(KhivaArray array)
Calculates the minimum value for each time series within array.

Return The minimum value of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.NumberCrossingM(KhivaArray array, int m)
Calculates the number of m-crossings. A m-crossing is defined as two sequential values where the first value is lower than m and the next is greater, or viceversa.If you set m to zero, you will get the number of zero crossings.

Return The number of m-crossings of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `m`: The m value.

static KhivaArray Khiva.Features.NumberCwtPeaks(KhivaArray array, int maxW)
This feature calculator searches for different peaks. To do so, the time series is smoothed by a ricker wavelet and for widths ranging from 1 to max_w.This feature calculator returns the number of peaks that occur at enough width scales and with sufficiently high Signal-to-Noise-Ratio (SNR).

Return The number of peaks for each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `maxW`: The maximum width to consider.

static KhivaArray Khiva.Features.NumberPeaks(KhivaArray array, int n)
Calculates the number of peaks of at least support n in the time series $array$. A peak of support n is defined as a subsequence of $array$ where a value occurs, which is bigger than its n neighbours to the left and to the right.

Return The number of peaks of at least support n.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `n`: The support of the peak.

static KhivaArray Khiva.Features.PartialAutocorrelation(KhivaArray array, KhivaArray la
Calculates the value of the partial autocorrelation function at the given lag. The lag k partial autocorrelation of a time series $\{x_t, t = 1 \dots T\}$ equals the partial correlation of x_t and x_{t-k}, adjusted for the intermediate variables $\{x_{t-1}, \dots, x_{t-k+1}\}$ ([1]). Following[2], it can be defined as:

$$\alpha_k = \frac{Cov(x_t, x_{t-k}|x_{t-1}, \ldots, x_{t-k+1})}{\sqrt{Var(x_t|x_{t-1}, \ldots, x_{t-k+1})Var(x_{t-k}|x_{t-1}, \ldots, x_{t-k+1})}}$$

with(a) $x_t = f(x_{t-1}, \ldots, x_{t-k+1})$ and(b) $x_{t-k} = f(x_{t-1}, \ldots, x_{t-k+1})$ being AR(k-1) models that can be fitted by OLS.Be aware that in (a), the regression is done on past values to predict x_t whereas in (b), future values are used to calculate the past value x_{t-k} . It is said in [1] that "for an AR(p), the partial autocorrelations \form#21 will be nonzero for \form#22

and zero for \form#23." With this property, it is used to determine the lag of an AR-Process.

[1] Box, G.E., Jenkins, G.M., Reinsel, G.C., & Ljung, G.M. (2015). Time series analysis: forecasting and control. John Wiley & Sons. [2] https://onlinecourses.science.psu.edu/stat510/node/62

Return Returns partial autocorrelation for each time series for the given lag.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `lags`: Indicates the lags to be calculated.

static KhivaArray Khiva.Features.PercentageOfReoccurringDatapointsToAllDatapoints(Khiva

Calculates the percentage of unique values, that are present in the time series more than once.

$$len(differentvaluesoccurringmorethanonce)/len(differentvalues)$$

This means the percentage is normalized to the number of unique values, in contrast to the percentageOfReoccurringValuesToAllValues.

Return Returns the percentage of unique values, that are present in the time series more than once.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `isSorted`: Indicates if the input time series is sorted or not. Defaults to false.

static KhivaArray Khiva.Features.PercentageOfReoccurringValuesToAllValues(KhivaArray a

Calculates the percentage of unique values, that are present in the time series more than once.

$$\frac{\textit{number of data points occurring more than once}}{\textit{number of all data points}})$$

This means the percentage is normalized to the number of unique values, in contrast to the percentageOfReoccurringDatapointsToAllDatapoints.

Return Returns the percentage of unique values, that are present in the time series more than once.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `isSorted`: Indicates if the input time series is sorted or not. Defaults to false.

static KhivaArray Khiva.Features.Quantile(KhivaArray array, KhivaArray q, float precis

Returns values at the given quantile.

Return Values at the given quantile.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `q`: Percentile(s) at which to extract score(s). One or many.
- `precision`: Number of decimals expected. Defaults to 1e8F.

static KhivaArray Khiva.Features.RangeCount(KhivaArray array, float min, float max)
Counts observed values within the interval [min, max).

Return Values at the given range.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `min`: Value that sets the lower limit.
- `max`: Value that sets the upper limit.

static KhivaArray Khiva.Features.RatioBeyondRSigma(KhivaArray array, float r)
Calculates the ratio of values that are more than $r * std(x)$ (so r sigma) away from the mean of x.

Return The ratio of values that are more than $r * std(x)$ (so r sigma) away from the mean of x.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `r`: Number of times that the values should be away from.

static KhivaArray Khiva.Features.RatioValueNumberToTimeSeriesLength(KhivaArray array)
Calculates a factor which is 1 if all values in the time series occur only once, and below one if this is not the case. In principle, it just returns:

$$\frac{number\_unique\_values}{number\_values}$$

Return The ratio of unique values with respect to the total number of values.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.SampleEntropy(KhivaArray array)
Calculates a vectorized sample entropy algorithm. https://en.wikipedia.org/wiki/Sample_entropy https://www.ncbi.nlm.nih.gov/pubmed/10843903?dopt=Abstract For short time-series this method is highly dependent on the parameters, but should be stable for N > 2000, see: Yentes et al. (2012) - The Appropriate Use of Approximate Entropy and Sample Entropy with Short Data Sets Other shortcomings and alternatives discussed in: Richman & Moorman (2000) - Physiological time-series analysis using approximate entropy and sample entropy.

Return An array with the same dimensions as array, whose values (time series in dimension 0) contains the vectorized sample entropy for all the input time series in array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.Skewness(KhivaArray array)

Calculates the sample skewness of array (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).

Return *KhivaArray* containing the skewness of each time series in array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.SpktWelchDensity(KhivaArray array, int coeff)

Estimates the cross power spectral density of the time series array at different frequencies. To do so, the time series is first shifted from the time domain to the frequency domain.

Welch's method computes an estimate of the power spectral density by dividing the data into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms. [1] P.Welch, "The use of the fast Fourier transform for the estimation of power spectra: A method based on time

averaging over short, modified periodograms", IEEE Trans. Audio Electroacoust. vol. 15, pp. 70-73, 1967. [2] M.S.Bartlett, "Periodogram Analysis and Continuous Spectra", Biometrika, vol. 37, pp. 1-16, 1950. [3] Rabiner, Lawrence R., and B. Gold. "Theory and Application of Digital Signal Processing" Prentice-Hall, pp. 414-419, 1975.

Return *KhivaArray* containing the power spectrum of the different frequencies for each time series in array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `coeff`: The coefficient to be returned.

static KhivaArray Khiva.Features.StandardDeviation(KhivaArray array)

Calculates the standard deviation of each time series within array.

Return The standard deviation of each time series within array.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.SumOfReoccurringDatapoints(KhivaArray array, bool isS

Calculates the sum of all data points, that are present in the time series more than once.

Return Returns the sum of all data points, that are present in the time series more than once.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `isSorted`: Indicates if the input time series is sorted or not. Defaults to false.

static KhivaArray Khiva.Features.SumOfReoccurringValues(KhivaArray array, bool isSorte

Calculates the sum of all values, that are present in the time series more than once.

Return Returns the sum of all values, that are present in the time series more than once.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `isSorted`: Indicates if the input time series is sorted or not. Defaults to false.

static KhivaArray Khiva.Features.SumValues(KhivaArray array)
Calculates the sum over the time series array.

Return An array containing the sum of values in each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.SymmetryLooking(KhivaArray array, float r)
Calculates if the distribution of array *looks symmetric*. This is the case if

$$|mean(array) - median(array)| r * (max(array) - min(array))$$

Return An array denoting if the input time series look symmetric.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `r`: The percentage of the range to compare with.

static KhivaArray Khiva.Features.TimeReversalAsymmetryStatistic(KhivaArray array, int
This function calculates the value of:

$$\frac{1}{n - 2lag} \sum_{i=0}^{n-2lag} x_{i+2\cdot lag}^2 \cdot x_{i+lag} - x_{i+lag} \cdot x_i^2$$

which is

$$\mathbb{E}[L^2(X)^2 \cdot L(X) - L(X) \cdot X^2]$$

where $\mathbb{E}$ is the mean and L is the lag operator. It was proposed in [1] as a promising feature to extract from time series.

[1] Fulcher, B.D., Jones, N.S. (2014). Highly comparative feature-based time-series classification. Knowledge and Data Engineering, IEEE Transactions on 26, 3026–3037.

Return An array containing the time reversal asymmetry statistic value in each time series.

Parameters

- `array`: xpects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `lag`: The lag to be computed.

static KhivaArray Khiva.Features.ValueCount(KhivaArray array, float v)
Counts occurrences of value in the time series array.

Return An array containing the count of the given value in each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `v`: The value to be counted.

static KhivaArray Khiva.Features.Variance(KhivaArray array)
Computes the variance for the time series array.

Return An array containing the variance in each time series.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Features.VarianceLargerThanStandardDeviation(KhivaArray array)
Calculates if the variance of array is greater than the standard deviation. In other words, if the variance of array is larger than 1.

Return An array denoting if the variance of array is greater than the standard deviation.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

1.6 Namespace Khiva

class Library
Class to change internal properties of the Khiva library.

Public Types

enum Backend
Khiva Backend.

Values:

KhivaBackendDefault = 0
DEFAULT Backend.

KhivaBackendCpu = 1
CPU Backend.

KhivaBackendCuda = 2
CUDA Backend.

KhivaBackendOpencl = 4
OPENCL Backend.

Property

property Khiva::Library::CurrentBackend
Getters and setters for the Khiva backend.

property Khiva::Library::SupportedBackends
Supported Khiva backends.

property Khiva::Library::BackendInfo
Information getter for the current backend.

property Khiva::Library::Device
Getter and setter for the Khiva device.

property Khiva::Library::DeviceCount
Getter for the device count.

property Khiva::Library::Version
Getter for the Khiva version.

Public Static Functions

static void Khiva.Library.PrintBackendInfo()
Prints information of the current backend.

1.7 Namespace LinAlg

class LinAlg
Khiva Linear Algebra class containing linear algebra methods.

Public Static Functions

static KhivaArray Khiva.LinAlg.Lls(KhivaArray a, KhivaArray b)
Calculates the minimum norm least squares solution x ($\|Ax - b\|^2$) to $Ax = b$. This function uses the singular value decomposition function of Arrayfire.The actual formula that this function computes is $x = VD\dagger U^T b$. Where U and V are orthogonal matrices and $D\dagger$ contains the inverse values of the singular values contained in D if they are not zero, and zero otherwise.

Return Contains the solution to the linear equation problem minimizing the norm 2.

Parameters

- `a`: A coefficient matrix containing the coefficients of the linear equation problem to solve.
- `b`: A vector with the measured values.

1.8 Namespace Matrix

class Matrix
Khiva *Matrix* Profile class containing matrix profile methods.

Public Static Functions

static Tuple<KhivaArray, KhivaArray, KhivaArray> Khiva.Matrix.FindBestNDiscords(KhivaA
Primitive of the findBestNDiscords function.

Return Tuple with the distance of the best N discords, the indices of the best N discords and the indices of the query sequences that produced the "N" bigger discords.

Parameters

- `profile`: The matrix profile containing the minimum distance of each subsequence.
- `index`: The matrix profile index containing the index of the most similar subsequence.
- `m`: Length of the matrix profile.
- `n`: Number of discords to extract.
- `selfJoin`: Indicates whether the input profile comes from a self join operation or not. It determines whether the mirror similar region is included in the output or not.

static Tuple<KhivaArray, KhivaArray, KhivaArray> Khiva.Matrix.FindBestNMotifs(KhivaArr

Primitive of the findBestNMotifs function.

Return Tuple with the distance of the best N motifs, the indices of the best N motifs, the indices of the query sequences that produced the minimum reported in the motifs.

Parameters

- `profile`: The matrix profile containing the minimum distance of each subsequence.
- `index`: The matrix profile index containing where each minimum occurs.
- `m`: Subsequence length value used to calculate the input matrix profile.
- `n`: Number of motifs to extract.
- `selfJoin`: Indicates whether the input profile comes from a self join operation or not. It determines whether the mirror similar region is included in the output or not.

static Tuple<KhivaArray, KhivaArray> Khiva.Matrix.Stomp(KhivaArray tssa, KhivaArray ts

Primitive of the STOMP algorithm.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). *Matrix* Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Return Tuple with the matrix profile, which has the distance to the closer element of the subsequence from 'tssa' in 'tssb' and the matrix profile index, which points to where the aforementioned minimum is located.

Parameters

- `tssa`: Query time series.
- `tssb`: Reference time series.
- `m`: Pointer to a long with the length of the subsequence.

static Tuple<KhivaArray, KhivaArray> Khiva.Matrix.StompSelfJoin(KhivaArray tss, long m

Primitive of the STOMP self join algorithm.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). *Matrix* Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Return Tuple with the matrix profile, which has the distance to the closer element of the subsequence from 'tss' in a different location of itself and the matrix profile index, which points to where the aforementioned minimum is located.

Parameters

- `tss`: Query and reference time series.
- `m`: Pointer to a long with the length of the subsequence.

static KhivaArray Khiva.Matrix.Mass(KhivaArray query, KhivaArray tss)

Mueen's Algorithm for Similarity Search.

The result has the following structure:

- 1st dimension corresponds to the index of the subsequence in the time series.
- 2nd dimension corresponds to the number of queries.
- 3rd dimension corresponds to the number of time series.

For example, the distance in the position (1, 2, 3) correspond to the distance of the third query to the fourth time series for the second subsequence in the time series.

[1] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk and Eamonn Keogh (2016). *Matrix* Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. IEEE ICDM 2016.

Return Resulting distances.

Parameters

- `query`: Array whose first dimension is the length of the query time series and the second dimension is the number of queries.
- `tss`: Array whose first dimension is the length of the time series and the second dimension is the number of time series.

static Tuple<KhivaArray, KhivaArray> Khiva.Matrix.FindBestNOccurrences(KhivaArray quer

Calculates the N best matches of several queries in several time series. The result has the following structure:

- 1st dimension corresponds to the nth best match.
- 2nd dimension corresponds to the number of queries.
- 3rd dimension corresponds to the number of time series.

For example, the distance in the position (1, 2, 3) corresponds to the second best distance of the third query in the fourth time series. The index in the position (1, 2, 3) is the is the index of the subsequence which leads to the second best distance of the third query in the fourth time series.

Return Tuple with the resulting distances and indexes.

Parameters

- `query`: Array whose first dimension is the length of the query time series and the second dimension is the number of queries.
- `tss`: Array whose first dimension is the length of the time series and the second dimension is the number of time series.
- `n`: Number of matches to return.

1.9 Namespace Normalization

class Normalization

Khiva *Normalization* class containing several normalization methods.

Public Static Functions

static KhivaArray Khiva.Normalization.DecimalScalingNorm(KhivaArray tss)

Normalizes the given time series according to its maximum value and adjusts each value within the range (-1, 1).

Return An array with the same dimensions as tss, whose values (time series in dimension 0) have been normalized by dividing each number by 10^j, where j is the number of integer digits of the max number in the time series.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static void Khiva.Normalization.DecimalScalingNorm(ref KhivaArray tss)

Same as decimal_scaling_norm, but it performs the operation inplace, without allocating further memory.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Normalization.MaxMinNorm(KhivaArray tss, double high, double l

Normalizes the given time series according to its minimum and maximum value and adjusts each value within the range[low, high].

Return *KhivaArray* with the same dimensions as tss, whose values (time series in dimension 0) have been normalized by maximum and minimum values, and scaled as per high and low parameters.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `high`: Maximum final value (Defaults to 1.0).
- `low`: Minimum final value (Defaults to 0.0).
- `epsilon`: Safeguard for constant (or near constant) time series as the operation implies a unit scale operation between min and max values in the tss.

static void Khiva.Normalization.MaxMinNorm(ref KhivaArray tss, double high, double lo

Same as max_min_norm, but it performs the operation inplace, without allocating further memory.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `high`: Maximum final value (Defaults to 1.0).
- `low`: Minimum final value (Defaults to 0.0).

- `epsilon`: Safeguard for constant (or near constant) time series as the operation implies a unit scale operation between min and max values in the tss.

static KhivaArray Khiva.Normalization.MeanNorm(KhivaArray tss)
Normalizes the given time series according to its maximum-minimum value and its mean. It follows the following formulae:

$$\acute{x} = \frac{x - mean(x)}{max(x) - min(x)}.$$

Return An array with the same dimensions as tss, whose values (time series in dimension 0) have been normalized by subtracting the mean from each number and dividing each number by $max(x) - min(x)$, in the time series.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static void Khiva.Normalization.MeanNorm(ref KhivaArray tss)
Normalizes the given time series according to its maximum-minimum value and its mean. It follows the following formulae:

$$\acute{x} = \frac{x - mean(x)}{max(x) - min(x)}.$$

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Normalization.ZNorm(KhivaArray tss, double epsilon)
Calculates a new set of times series with zero mean and standard deviation one.

Return *KhivaArray* with the same dimensions as tss where the time series have been adjusted for zero mean and one as standard deviation.

Parameters

- `tss`: Time series concatenated in a single row.
- `epsilon`: Minimum standard deviation to consider. It acts as a gatekeeper for those time series that may be constant or near constant.

static void Khiva.Normalization.ZNorm(ref KhivaArray tss, double epsilon)
Adjusts the time series in the given input and performs z-norm inplace(without allocating further memory).

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `epsilon`: Minimum standard deviation to consider. It acts as a gatekeeper for those time series that may be constant or near constant.

1.10 Namespace Polynomial

class Polynomial
Khiva *Polynomial* class containing a number of polynomial methods.

Public Static Functions

static KhivaArray Khiva.Polynomial.PolyFit(KhivaArray x, KhivaArray y, int deg)
Least squares polynomial fit. Fit a polynomial $p(x) = p[0] * x^{deg} + ... + p[deg]$ of degree deg to points (x, y). Returns a vector of coefficients p that minimises the squared error.

Return *Polynomial* coefficients, highest power first.

Parameters

- `x`: x-coordinates of the M sample points $(x[i], y[i])$.
- `y`: y-coordinates of the sample points.
- `deg`: Degree of the fitting polynomial.

static KhivaArray Khiva.Polynomial.Roots(KhivaArray p)
Calculates the roots of a polynomial with coefficients given in p. The values in the rank-1 array p are coefficients of a polynomial.If the length of p is $n + 1$ then the polynomial is described by:

$$p[0] * x^n + p[1] * x^{n-1} + ... + p[n-1] * x + p[n]$$

Return *KhivaArray* containing the roots of the polynomial.

Parameters

- `p`: *KhivaArray* of polynomial coefficients.

1.11 Namespace Regression

class Regression
Khiva *Regression* class containing regression methods.

Public Static Functions

static Tuple<KhivaArray, KhivaArray, KhivaArray, KhivaArray, KhivaArray> Khiva.Regress
Calculates a linear least-squares regression for two sets of measurements. Both arrays should have the same length.

Return Tuple with the slope of the regression line, the correlation coefficient, the two-sided p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic and the standard error of the estimated gradient.

Parameters

- `xss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `yss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

1.12 Namespace Regularization

class Regularization
Khiva *Regularization* class containing different regularization methods.

Public Static Functions

static KhivaArray Khiva.Regularization.GroupBy(KhivaArray array, int aggregationFunctio
Group by operation in the input array using n_columns_key columns as group keys and n_columns_value columns as values.The data is expected to be sorted.The aggregation function determines the operation to aggregate the values.

Parameters

- `array`: Expects an input array whose dimension zero is the length of the time series(all the same) and dimension one indicates the number of time series.
- `aggregationFunction`: Function to be used in the aggregation. It receives an integer which indicates the function to be applied: { 0 : mean, 1 : median 2 : min, 3 : max, 4 : stdev, 5 : var, default : mean } /param>

  Return An array with the values of the group keys aggregated using the aggregation_function.

  Parameters

  - `nColumnsKey`: Number of columns conforming the key.
  - `nColumnsValue`: Number of columns conforming the value (they are expected to be consecutive to the column.

1.13 Namespace Statistics

class Statistics
Khiva *Statistics* class containing statistics methods.

Public Static Functions

static KhivaArray Khiva.Statistics.CovarianceStatistics(KhivaArray tss, bool unbiased)
Returns the covariance matrix of the time series contained in tss.

Return The covariance matrix of the time series.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `unbiased`: Determines whether it divides by n - 1 (if false) or n (if true).

static KhivaArray Khiva.Statistics.KurtosisStatistics(KhivaArray tss)
Returns the kurtosis of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).

Return The kurtosis of tss.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Statistics.LjungBox(KhivaArray tss, long lags)
The Ljung–Box test checks that data within the time series are independently distributed (i.e. the correlations in the population from which the sample is taken are 0, so that any observed correlations in the data

result from randomness of the sampling process). Data are no independently distributed, if they exhibit serial correlation.

The test statistic is:

$$Q = n\,(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k}$$

where ''n" is the sample size, $\hat{\rho}k$ is the sample autocorrelation at lag ''k'', and ''h" is the number of lags being tested.Under H_0 the statistic Q follows a $\chi^2(h)$. For significance level α, the $criticalregion$ for rejection of the hypothesis of randomness is:

$$Q > \chi^2_{1-\alpha,h}$$

where $\chi^2_{1-\alpha,h}$ is the α-quantile of the chi-squared distribution with ''h" degrees of freedom.

[1] G.M.Ljung G. E.P.Box (1978). On a measure of lack of fit in time series models. Biometrika, Volume 65, Issue 2, 1 August 1978, Pages 297–303.

Return The Ljung-Box statistic test.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `lags`: Number of lags being tested.

static KhivaArray Khiva.Statistics.MomentStatistics(KhivaArray tss, int k)
Returns the kth moment of the given time series.

Return The kth moment of the given time series.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.
- `k`: The specific moment to be calculated.

static KhivaArray Khiva.Statistics.QuantileStatistics(KhivaArray tss, KhivaArray q, fl
Returns values at the given quantile.

Return Values at the given quantile.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.NOTE: the time series should be sorted.
- `q`: Percentile(s) at which to extract score(s). One or many.
- `precision`: Number of decimals expected.

static KhivaArray Khiva.Statistics.QuantilesCutStatistics(KhivaArray tss, float quanti
Discretizes the time series into equal-sized buckets based on sample quantiles.

Return *Matrix* with the categories, one category per row, the start of the category in the first column and the end in the second category.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.NOTE: the time series should be sorted.
- `quantiles`: Number of quantiles to extract. From 0 to 1, step 1/quantiles.
- `precision`: Number of decimals expected.

static KhivaArray Khiva.Statistics.SampleStdevStatistics(KhivaArray tss)

Estimates standard deviation based on a sample. The standard deviation is calculated using the "n-1" method.

Return The sample standard deviation.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

static KhivaArray Khiva.Statistics.SkewnessStatistics(KhivaArray tss)

Calculates the sample skewness of tss (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).

Return *KhivaArray* containing the skewness of each time series in tss.

Parameters

- `tss`: Expects an input array whose dimension zero is the length of the time series (all the same) and dimension one indicates the number of time series.

- *Namespace Array*
- *Namespace Clustering*
- *Namespace Dimensionality*
- *Namespace Distances*
- *Namespace Features*
- *Namespace Khiva*
- *Namespace LinAlg*
- *Namespace Matrix*
- *Namespace Normalization*
- *Namespace Polynomial*
- *Namespace Regression*
- *Namespace Regularization*
- *Namespace Statistics*

CHAPTER 2

FAQ

1. **What are the Khiva Array restrictions?**

   Khiva Arrays can have up to 4 dimensions, each one needs to have the same length and the same type.

2. **Can I use the Khiva package without having a GPU for computation?**

   Of course. It is right that Khiva algorithms are designed to run on GPU, but a CPU backend could be set.

3. **What OS are supported?**

   Nowadays, Khiva is supported on Windows, Linux and MacOS.

CHAPTER 3

License

```
Mozilla Public License, version 2.0

1. Definitions

1.1. "Contributor"

     means each individual or legal entity that creates, contributes to the
     creation of, or owns Covered Software.

1.2. "Contributor Version"

     means the combination of the Contributions of others (if any) used by a
     Contributor and that particular Contributor's Contribution.

1.3. "Contribution"

     means Covered Software of a particular Contributor.

1.4. "Covered Software"

     means Source Code Form to which the initial Contributor has attached the
     notice in Exhibit A, the Executable Form of such Source Code Form, and
     Modifications of such Source Code Form, in each case including portions
     thereof.

1.5. "Incompatible With Secondary Licenses"
     means

     a. that the initial Contributor has attached the notice described in
        Exhibit B to the Covered Software; or

     b. that the Covered Software was made available under the terms of
        version 1.1 or earlier of the License, but not also under the terms of
        a Secondary License.
```

(continues on next page)

(continued from previous page)

```
1.6. "Executable Form"

     means any form of the work other than Source Code Form.

1.7. "Larger Work"

     means a work that combines Covered Software with other material, in a
     separate file or files, that is not Covered Software.

1.8. "License"

     means this document.

1.9. "Licensable"

     means having the right to grant, to the maximum extent possible, whether
     at the time of the initial grant or subsequently, any and all of the
     rights conveyed by this License.

1.10. "Modifications"

     means any of the following:

     a. any file in Source Code Form that results from an addition to,
        deletion from, or modification of the contents of Covered Software; or

     b. any new file in Source Code Form that contains any Covered Software.

1.11. "Patent Claims" of a Contributor

      means any patent claim(s), including without limitation, method,
      process, and apparatus claims, in any patent Licensable by such
      Contributor that would be infringed, but for the grant of the License,
      by the making, using, selling, offering for sale, having made, import,
      or transfer of either its Contributions or its Contributor Version.

1.12. "Secondary License"

      means either the GNU General Public License, Version 2.0, the GNU Lesser
      General Public License, Version 2.1, the GNU Affero General Public
      License, Version 3.0, or any later versions of those licenses.

1.13. "Source Code Form"

      means the form of the work preferred for making modifications.

1.14. "You" (or "Your")

      means an individual or a legal entity exercising rights under this
      License. For legal entities, "You" includes any entity that controls, is
      controlled by, or is under common control with You. For purposes of this
      definition, "control" means (a) the power, direct or indirect, to cause
      the direction or management of such entity, whether by contract or
      otherwise, or (b) ownership of more than fifty percent (50%) of the
      outstanding shares or beneficial ownership of such entity.
```

(continues on next page)

(continued from previous page)

```
2. License Grants and Conditions

2.1. Grants

     Each Contributor hereby grants You a world-wide, royalty-free,
     non-exclusive license:

     a. under intellectual property rights (other than patent or trademark)
        Licensable by such Contributor to use, reproduce, make available,
        modify, display, perform, distribute, and otherwise exploit its
        Contributions, either on an unmodified basis, with Modifications, or
        as part of a Larger Work; and

     b. under Patent Claims of such Contributor to make, use, sell, offer for
        sale, have made, import, and otherwise transfer either its
        Contributions or its Contributor Version.

2.2. Effective Date

     The licenses granted in Section 2.1 with respect to any Contribution
     become effective for each Contribution on the date the Contributor first
     distributes such Contribution.

2.3. Limitations on Grant Scope

     The licenses granted in this Section 2 are the only rights granted under
     this License. No additional rights or licenses will be implied from the
     distribution or licensing of Covered Software under this License.
     Notwithstanding Section 2.1(b) above, no patent license is granted by a
     Contributor:

     a. for any code that a Contributor has removed from Covered Software; or

     b. for infringements caused by: (i) Your and any other third party's
        modifications of Covered Software, or (ii) the combination of its
        Contributions with other software (except as part of its Contributor
        Version); or

     c. under Patent Claims infringed by Covered Software in the absence of
        its Contributions.

     This License does not grant any rights in the trademarks, service marks,
     or logos of any Contributor (except as may be necessary to comply with
     the notice requirements in Section 3.4).

2.4. Subsequent Licenses

     No Contributor makes additional grants as a result of Your choice to
     distribute the Covered Software under a subsequent version of this
     License (see Section 10.2) or under the terms of a Secondary License (if
     permitted under the terms of Section 3.3).

2.5. Representation

     Each Contributor represents that the Contributor believes its
     Contributions are its original creation(s) or it has sufficient rights to
```

(continues on next page)

(continued from previous page)

```
     grant the rights to its Contributions conveyed by this License.

2.6. Fair Use

     This License is not intended to limit any rights You have under
     applicable copyright doctrines of fair use, fair dealing, or other
     equivalents.

2.7. Conditions

     Sections 3.1, 3.2, 3.3, and 3.4 are conditions of the licenses granted in
     Section 2.1.


3. Responsibilities

3.1. Distribution of Source Form

     All distribution of Covered Software in Source Code Form, including any
     Modifications that You create or to which You contribute, must be under
     the terms of this License. You must inform recipients that the Source
     Code Form of the Covered Software is governed by the terms of this
     License, and how they can obtain a copy of this License. You may not
     attempt to alter or restrict the recipients' rights in the Source Code
     Form.

3.2. Distribution of Executable Form

     If You distribute Covered Software in Executable Form then:

     a. such Covered Software must also be made available in Source Code Form,
        as described in Section 3.1, and You must inform recipients of the
        Executable Form how they can obtain a copy of such Source Code Form by
        reasonable means in a timely manner, at a charge no more than the cost
        of distribution to the recipient; and

     b. You may distribute such Executable Form under the terms of this
        License, or sublicense it under different terms, provided that the
        license for the Executable Form does not attempt to limit or alter the
        recipients' rights in the Source Code Form under this License.

3.3. Distribution of a Larger Work

     You may create and distribute a Larger Work under terms of Your choice,
     provided that You also comply with the requirements of this License for
     the Covered Software. If the Larger Work is a combination of Covered
     Software with a work governed by one or more Secondary Licenses, and the
     Covered Software is not Incompatible With Secondary Licenses, this
     License permits You to additionally distribute such Covered Software
     under the terms of such Secondary License(s), so that the recipient of
     the Larger Work may, at their option, further distribute the Covered
     Software under the terms of either this License or such Secondary
     License(s).

3.4. Notices

     You may not remove or alter the substance of any license notices
```

(continues on next page)

(continued from previous page)

```
     (including copyright notices, patent notices, disclaimers of warranty, or
     limitations of liability) contained within the Source Code Form of the
     Covered Software, except that You may alter any license notices to the
     extent required to remedy known factual inaccuracies.

3.5. Application of Additional Terms

     You may choose to offer, and to charge a fee for, warranty, support,
     indemnity or liability obligations to one or more recipients of Covered
     Software. However, You may do so only on Your own behalf, and not on
     behalf of any Contributor. You must make it absolutely clear that any
     such warranty, support, indemnity, or liability obligation is offered by
     You alone, and You hereby agree to indemnify every Contributor for any
     liability incurred by such Contributor as a result of warranty, support,
     indemnity or liability terms You offer. You may include additional
     disclaimers of warranty and limitations of liability specific to any
     jurisdiction.

4. Inability to Comply Due to Statute or Regulation

   If it is impossible for You to comply with any of the terms of this License
   with respect to some or all of the Covered Software due to statute,
   judicial order, or regulation then You must: (a) comply with the terms of
   this License to the maximum extent possible; and (b) describe the
   limitations and the code they affect. Such description must be placed in a
   text file included with all distributions of the Covered Software under
   this License. Except to the extent prohibited by statute or regulation,
   such description must be sufficiently detailed for a recipient of ordinary
   skill to be able to understand it.

5. Termination

5.1. The rights granted under this License will terminate automatically if You
     fail to comply with any of its terms. However, if You become compliant,
     then the rights granted under this License from a particular Contributor
     are reinstated (a) provisionally, unless and until such Contributor
     explicitly and finally terminates Your grants, and (b) on an ongoing
     basis, if such Contributor fails to notify You of the non-compliance by
     some reasonable means prior to 60 days after You have come back into
     compliance. Moreover, Your grants from a particular Contributor are
     reinstated on an ongoing basis if such Contributor notifies You of the
     non-compliance by some reasonable means, this is the first time You have
     received notice of non-compliance with this License from such
     Contributor, and You become compliant prior to 30 days after Your receipt
     of the notice.

5.2. If You initiate litigation against any entity by asserting a patent
     infringement claim (excluding declaratory judgment actions,
     counter-claims, and cross-claims) alleging that a Contributor Version
     directly or indirectly infringes any patent, then the rights granted to
     You by any and all Contributors for the Covered Software under Section
     2.1 of this License shall terminate.

5.3. In the event of termination under Sections 5.1 or 5.2 above, all end user
     license agreements (excluding distributors and resellers) which have been
     validly granted by You or Your distributors under this License prior to
     termination shall survive termination.
```

(continues on next page)

(continued from previous page)

```
6. Disclaimer of Warranty

   Covered Software is provided under this License on an "as is" basis,
   without warranty of any kind, either expressed, implied, or statutory,
   including, without limitation, warranties that the Covered Software is free
   of defects, merchantable, fit for a particular purpose or non-infringing.
   The entire risk as to the quality and performance of the Covered Software
   is with You. Should any Covered Software prove defective in any respect,
   You (not any Contributor) assume the cost of any necessary servicing,
   repair, or correction. This disclaimer of warranty constitutes an essential
   part of this License. No use of  any Covered Software is authorized under
   this License except under this disclaimer.

7. Limitation of Liability

   Under no circumstances and under no legal theory, whether tort (including
   negligence), contract, or otherwise, shall any Contributor, or anyone who
   distributes Covered Software as permitted above, be liable to You for any
   direct, indirect, special, incidental, or consequential damages of any
   character including, without limitation, damages for lost profits, loss of
   goodwill, work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses, even if such party shall have been
   informed of the possibility of such damages. This limitation of liability
   shall not apply to liability for death or personal injury resulting from
   such party's negligence to the extent applicable law prohibits such
   limitation. Some jurisdictions do not allow the exclusion or limitation of
   incidental or consequential damages, so this exclusion and limitation may
   not apply to You.

8. Litigation

   Any litigation relating to this License may be brought only in the courts
   of a jurisdiction where the defendant maintains its principal place of
   business and such litigation shall be governed by laws of that
   jurisdiction, without reference to its conflict-of-law provisions. Nothing
   in this Section shall prevent a party's ability to bring cross-claims or
   counter-claims.

9. Miscellaneous

   This License represents the complete agreement concerning the subject
   matter hereof. If any provision of this License is held to be
   unenforceable, such provision shall be reformed only to the extent
   necessary to make it enforceable. Any law or regulation which provides that
   the language of a contract shall be construed against the drafter shall not
   be used to construe this License against a Contributor.


10. Versions of the License

10.1. New Versions

      Mozilla Foundation is the license steward. Except as provided in Section
      10.3, no one other than the license steward has the right to modify or
      publish new versions of this License. Each version will be given a
      distinguishing version number.
```

(continues on next page)

(continued from previous page)

```
10.2. Effect of New Versions

      You may distribute the Covered Software under the terms of the version
      of the License under which You originally received the Covered Software,
      or under the terms of any subsequent version published by the license
      steward.

10.3. Modified Versions

      If you create software not governed by this License, and you want to
      create a new license for such software, you may create and use a
      modified version of this License if you rename the license and remove
      any references to the name of the license steward (except to note that
      such modified license differs from this License).

10.4. Distributing Source Code Form that is Incompatible With Secondary
      Licenses If You choose to distribute Source Code Form that is
      Incompatible With Secondary Licenses under the terms of this version of
      the License, the notice described in Exhibit B of this License must be
      attached.

Exhibit A - Source Code Form License Notice

      This Source Code Form is subject to the
      terms of the Mozilla Public License, v.
      2.0. If a copy of the MPL was not
      distributed with this file, You can
      obtain one at
      http://mozilla.org/MPL/2.0/.

If it is not possible or desirable to put the notice in a particular file,
then You may include the notice in a location (such as a LICENSE file in a
relevant directory) where a recipient would be likely to look for such a
notice.

You may add additional accurate notices of copyright ownership.

Exhibit B - "Incompatible With Secondary Licenses" Notice

      This Source Code Form is "Incompatible
      With Secondary Licenses", as defined by
      the Mozilla Public License, v. 2.0.
```

CHAPTER 4

AUTHORS

4.1 Core Development Team

- Justo Ruiz-Ferrer (justo.ruiz@shapelets.io)
- Antonio Vilches (antonio.vilches@shapelets.io)
- Oscar Torreno (oscar.torreno@shapelets.io)
- David Cuesta (david.cuesta@shapelets.io)

4.2 Contributions

- Luis Sanchez (luis.sanchez@shapelets.io)

CHAPTER 5

How to contribute

We have just started! Our aim is to make the Khiva library the reference library for time series analysis in the fastest fashion. To achieve this target we need your help!

All contributions, bug reports, bug fixes, documentation improvements, enhancements and ideas are welcome. If you want to add one or two interesting feature calculators, implement a new feature selection process or just fix 1-2 typos, your help is appreciated.

If you want to help, just create a pull request on our github page.

5.1 Guidelines

5.1.1 Branching model

Our branching model has one permanent branch, **master**. We aim at using *master* as the main branch, where all features are merged. In this sense, we also use the master branch to contain the release versions of the Python Khiva library by means of git tags.

5.1.2 Contribution process

In order to contribute to the code base, we follow the next process:

1. The main branch is *master*, every developer should pull the current status of the branch before stating to develop any new feature. *git pull*

2. Create a new branch with the following pattern "feature/[name_of_the_feature]" *git checkout -b feature/example_feature*

3. Develop the new feature on the the new branch. It includes testing and documentation. *git commit -a -m "Bla, Bla, Bla"*

git push

4. Open a Pull Request to merge the feature branch in to *master*. Currently, a pull request has to be reviewed at least by one person.

5. Finally, delete the feature branch.

6. Move back to *master* branch. *git checkout master*

7. Pull the latest changes. *git pull*

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index

K