Package ‘analysisPipelines’

June 12, 2020

Type Package
Date 2020-06-12
Title Compose Interoperable Analysis Pipelines & Put Them in Production
Version 1.0.2
Description Enables data scientists to compose pipelines of analysis which consist of data manipulation, exploratory analysis & reporting, as well as modeling steps. Data scientists can use tools of their choice through an R interface, and compose interoperable pipelines between R, Spark, and Python. Credits to Mu Sigma for supporting the development of the package.

Note - To enable pipelines involving Spark tasks, the package uses the ‘SparkR’ package. The SparkR package needs to be installed to use Spark as an engine within a pipeline. SparkR is distributed natively with Apache Spark and is not distributed on CRAN. The SparkR version needs to directly map to the Spark version (hence the native distribution), and care needs to be taken to ensure that this is configured properly.

To install SparkR from Github, run the following command if you know the Spark version: ‘devtools::install_github('apache/spark@v2.x.x', subdir='R/pkg’)’.

The other option is to install SparkR by running the following terminal commands if Spark has already been installed: '$ export SPARK_HOME=/path/to/spark/directory && cd $SPARK_HOME/R/lib/SparkR/ && R -e "devtools::install("")"'.

Depends R (>= 3.4.0), magrittr, pipeR, methods
Imports ggplot2, dplyr, futile.logger, RCurl, rlang (>= 0.3.0), proto, purrr
Suggests plotly, knitr, rmarkdown, parallel, visNetwork, rjson, DT, shiny, R.devices, corrplot, car, foreign
Enhances SparkR, reticulate

BugReports https://github.com/Mu-Sigma/analysis-pipelines/issues

URL https://github.com/Mu-Sigma/analysis-pipelines

Encoding UTF-8
License Apache License 2.0
LazyLoad yes
LazyData yes

RoxygenNote 6.1.1

VignetteBuilder knitr

Collate 'analysisPipelines_package.R' 'core-functions.R'
    'core-functions-batch.R' 'core-functions-meta-pipelines.R'
    'core-streaming-functions.R' 'r-batch-eda-utilities.R'
    'r-helper-utilites-python.R'
    'spark-structured-streaming-utilities.R' 'zzz.R'

NeedsCompilation no

Author Naren Srinivasan [aut],
    Zubin Dowlaty [aut],
    Sanjay [ctb],
    Neeratyoy Mallik [ctb],
    Anoop S [ctb],
    Mu Sigma, Inc. [cre]

Maintainer ``Mu Sigma, Inc." <ird.experiencelab@mu-sigma.com>

Repository CRAN

Date/Publication 2020-06-12 08:00:02 UTC

R topics documented:

  AnalysisPipeline-class .......................................... 3
  analysisPipelines .................................................. 4
  assessEngineSetUp ............................................... 4
  BaseAnalysisPipeline-class ...................................... 5
  bivarPlots .......................................................... 6
  castKafkaStreamAsString .......................................... 7
  CheckColumnType .................................................. 8
  checkSchemaMatch ................................................ 8
  convertKafkaValueFromJson ....................................... 9
  correlationMatPlot .............................................. 10
  createPipelineInstance .......................................... 11
  exportAsMetaPipeline ............................................ 12
  generateOutput ................................................... 13
  generateReport ................................................... 14
  genericPipelineException ....................................... 15
  getDatatype ....................................................... 16
  getFeaturesForPyClassification .................................. 16
  getInput ........................................................... 17
  getILoggerDetails ................................................. 18
  getOutputById ..................................................... 19
  getPipeline ......................................................... 20
  getPipelinePrototype ............................................. 21
  getRegistry ........................................................ 22
  getResponse ........................................................ 23
## AnalysisPipeline-class

Class for constructing Analysis Pipelines for batch/one-time analyses

### Description

Class for constructing Analysis Pipelines for batch/one-time analyses

### Details

Inherits the base class `BaseAnalysisPipeline` class which holds the metadata including the registry of available functions, the data on which the pipeline is to be applied, as well as the pipeline itself.

Additionally, this class is meant to be used for batch/one-time processing. Contains additional slots to hold the data frame to be used for the pipeline and associated schema.

### Slots

- **input**: The input dataset on which analysis is to be performed
- **originalSchemaDf**: Empty data frame representing the schema of the input

---

### Index

- `AnalysisPipeline-class`
- `BaseAnalysisPipeline-class`
- `BasePipeline-class`
- `MetaAnalysisPipeline-class`
- `StreamingAnalysisPipeline-class`
- `loadMetaPipeline`
- `loadPipeline`
- `loadPredefinedFunctionRegistry`
- `loadRegistry`
- `prepExecution`
- `registerFunction`
- `savePipeline`
- `saveRegistry`
- `setInput`
- `setLoggerDetails`
- `setPythonEnvir`
- `sparkRSessionCreateIfNotPresent`
- `updateObject`
- `visualizePipeline`
- `getTargetForPyClassification`
- `getTerm`
- `ignoreCols`
- `isDependencyParam`
- `multiVarOutlierPlot`
- `outlierPlot`
- `univarCatDistPlots`
- `updateObject`
- `visualizePipeline`
The package aims at enabling data scientists to compose pipelines of analysis which consist of data manipulation, exploratory analysis & reporting, as well as modeling steps. It also aims to enable data scientists to use tools of their choice through an R interface, and compose interoperable pipelines between R, Spark, and Python. Credits to Mu Sigma for supporting the development of the package.

Note
To enable pipelines involving Spark tasks, the package uses the 'SparkR' package. Using Spark as an engine requires the SparkR package to be installed. SparkR is distributed natively with Apache Spark and is not distributed on CRAN. The SparkR version needs to directly map to the Spark version (hence the native distribution), and care needs to be taken to ensure that this is configured properly. To install from Github, run the following command, if you know the Spark version:

```r
devtools::install_github('apache/spark@v2.x.x', subdir='R/pkg')
```

The other option is to install SparkR by running the following terminal commands if Spark has already been installed:

```bash
$ export SPARK_HOME=/path/to/spark/directory
$ cd $SPARK_HOME/R/lib/SparkR/
$ R -e "devtools::install('.', )"
```

### Description
Assesses engine (R, Spark, Python, Spark Structured Streaming) set up

#### Usage

```r
assessEngineSetUp(object)
```

## S4 method for signature 'BaseAnalysisPipeline'

```r
assessEngineSetUp(object)
```
BaseAnalysisPipeline-class

Arguments

object A Pipeline object

Details

Assesses whether engines required for executing functions in an AnalysisPipeline or StreamingAnalysisPipeline object have been set up

This method is implemented on the base class as it is a shared functionality across Pipeline objects

Value

Tibble containing the details of available engines, whether they are required for a pipeline, a logical value reporting whether the engine has been set up, and comments.

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

```r
## Not run:
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>% univarCatDistPlots(uniCol = "Species", priColor = "blue", optionalPlots = 0) %>% assessEngineSetUp

## End(Not run)
```

BaseAnalysisPipeline-class

Description

Base class for AnalysisPipeline and StreamingAnalysisPipeline objects
bivarPlots

Details

The class which holds the metadata including the registry of available functions, the data on which the pipeline is to be applied, as well as the pipeline itself, and serves as the base class for various types of Pipeline objects such as Batch and Streaming.

This base class which contains the slots related to the registry, pipeline and output can be extended to create custom class for specific scenarios if required.

In the documentation, objects of classes which are subclasses of this class are referred to as 'Pipeline' objects

Slots

pipeline A tibble which holds functions to be called
pipelineExecutor A list containing details of the execution, such as topological ordering of functions to be executed, dependency map of functions, as well as logger configuration
output A list which holds all the functions output

See Also

Other Package core functions: MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getILoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

bivarPlots Bi-Variate Plot

Description

Bi-Variate Plot

Usage

bivarPlots(dataset, select_var_name_1, select_var_name_2, 
   priColor = "blue", secColor = "black")

Arguments

dataset the dataframe that needs to be loaded
select_var_name_1 the name of first column on which the plot needs to be generated
select_var_name_2 the name of second column on which the plot needs to be generated
priColor the primary color for the plots
secColor A secondary color for the plots
castKafkaStreamAsString

Details
A bivariate distribution graph on the selected columns from the dataframe. Selected two columns are on two axis' and a plot is generated.

Value
Bivariate plot

See Also
Other Package EDA Utilities functions: CheckColumnType, correlationMatPlot, getDatatype, ignoreCols, multiVarOutlierPlot, outlierPlot, univarCatDistPlots

Examples

bivarPlots(dataset = iris, select_var_name_1 = "Sepal.Length", select_var_name_2 = "Sepal.Width")

---

castKafkaStreamAsString

Connect to a Spark session

Description
Connect to a Spark session

Usage
castKafkaStreamAsString(streamObj)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>streamObj</td>
<td>Spark Structured Streaming DataFrame returned by read.stream function with source = 'kafka'</td>
</tr>
</tbody>
</table>

Details
Takes in a Structured Stream from Kafka created from read.stream(source = 'kafka',...) and returns a Structured Streaming DataFrame where the key and value from the Kafka stream are cast to string.

Value
Updated Spark Structured Streaming DataFrame with key, value, topic and timestamp from the Kafka stream

See Also
Other Spark utilities: convertKafkaValueFromJson, sparkRSessionCreateIfNotPresent
checkSchemaMatch

Description

Checks the schema of the input to a Pipeline object against the original

Usage

checkSchemaMatch(object, newData)

## S4 method for signature 'AnalysisPipeline'
checkSchemaMatch(object, newData)

CheckColumnType

Check for type of column

Description

Check for type of column

Usage

CheckColumnType(dataVector)

Arguments

dataVector a data vector of a column

Details

Checking for type of columns in the datavector

Value

column Type

See Also

Other Package EDA Utilites functions: bivarPlots, correlationMatPlot, getDatatype, ignoreCols, multiVarOutlierPlot, outlierPlot, univarCatDistPlots

Examples

CheckColumnType(iris$Sepal.Length)

checkSchemaMatch

Checks the schema of the input to a Pipeline object against the original

Description

Checks the schema of the input to a Pipeline object against the original

Usage

checkSchemaMatch(object, newData)

## S4 method for signature 'AnalysisPipeline'
checkSchemaMatch(object, newData)
**convertKafkaValueFromJson**

**Arguments**

- **object**: A Pipeline object
- **newData**: The newData that the pipeline is to be initialized with

**Details**

Checks the schema of the new data frame that the pipeline is to be initialized with against the original schema that the pipeline was saved with. Provides a detailed comparison

**Value**

Returns a list with details on added columns, removed columns, comparison between column classes, and a logical whether the schema has remained the same from the old dataframe to the new one

**See Also**

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLoggerDetails, getOutputByld, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

---

**convertKafkaValueFromJson**

*Connect to a Spark session*

**Description**

Connect to a Spark session

**Usage**

`convertKafkaValueFromJson(streamObj, schema)`

**Arguments**

- **streamObj**: Spark Structured Streaming DataFrame which is returned by the castKafkaStreamAsString function
- **schema**: A structType object created from SparkR specifying the schema of the json data present in the value attribute of the incoming Kafka stream

**Details**

Takes in a Structured Stream from Kafka created from `read.stream(source = 'kafka',...)` and returns a Structured Streaming DataFrame where the key and value from the Kafka stream are cast to string
correlationMatPlot

Value

Spark Structured Streaming DataFrame with the json data in the value attribute of the Kafka stream parsed into a DataFrame format

See Also

Other Spark utilities: castKafkaStreamAs STRING, sparkRSessionCreateIfNotPresent

correlationMatPlot  Correlation Matrix Plot

Description

A correlation matrix is created and plotted across all the columns in the dataset

Usage

correlationMatPlot(dataset, methodused = "everything")

Arguments

dataset the dataset that needs to be loaded
methodused methods to be used for computing correlation

Value

Correlation Matrix graph

See Also

Other Package EDA Utilites functions: CheckColumnType, bivarPlots, getDatatype, ignoreCols, multiVarOutlierPlot, outlierPlot, univarCatDistPlots

Examples

correlationMatPlot(dataset = iris)
createPipelineInstance

Create a Pipeline object from a meta-pipeline

Description

Create a Pipeline object from a meta-pipeline

Usage

createPipelineInstance(metaPipelineObj, newParams)

## S4 method for signature 'MetaAnalysisPipeline'
createPipelineInstance(metaPipelineObj, newParams)

Arguments

metaPipelineObj
  A MetaAnalysisPipeline object

newParams
  Either a nested named list containing all the functions in the pipeline, their arguments and corresponding values (OR) an object of class proto which is a pipeline prototype, with the new values of the arguments set. Refer the `getPipelinePrototype` method.

Details

This method instantiates a Pipeline object (both AnalysisPipeline and StreamingAnalysisPipeline) from a meta-pipeline as well as an object containing the new set of values for the arguments of all the functions in the pipeline.

Value

A Pipeline object

See Also

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `exportAsMetaPipeline`, `generateOutput`, `genericPipelineException`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPipeline`, `loadPredefinedFunctionRegistry`, `loadRegistry`, `prepExecution`, `registerFunction`, `savePipeline`, `saveRegistry`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`
**Examples**

```r
## Not run:
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>% univarCatDistPlots(uniCol = "Species") -> pipelineObj
pipelineObj %>% exportAsMetaPipeline -> exportedMetaPipeline
exportedMetaPipeline %>%
createPipelineInstance(newParams = exportedMetaPipeline %>%
getPipelinePrototype)

## End(Not run)
```

---

**exportAsMetaPipeline**  
Method to export a meta-pipeline

---

**Description**
Method to export a meta-pipeline

**Usage**

```r
exportAsMetaPipeline(object)
```

### S4 method for signature 'BaseAnalysisPipeline'

```r
exportAsMetaPipeline(object)
```

**Arguments**

- **object**  
  A Pipeline object

**Details**
This method exports a Pipeline object i.e. of the classes AnalysisPipeline or StreamingAnalysisPipeline as a meta-pipeline

**Value**

- an object of class "MetaAnalysisPipeline"

**See Also**

Other Package core functions:  
BaseAnalysisPipeline-class, MetaAnalysisPipeline-class,  
assessEngineSetUp, checkSchemaMatch, createPipelineInstance, generateOutput, genericPipelineException,  
getInput, getLoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry,  
initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline,  
loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline,  
saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline
Examples

```r
## Not run:
# ' pipelineObj <- AnalysisPipeline(input = iris)
 pipelineObj %>% univarCatDistPlots(uniCol = "Species") %>%
   exportAsMetaPipeline -> exportedMetaPipeline

## End(Not run)
```

---

**generateOutput**

Generate a list of outputs from Pipeline objects

**Description**

Generate a list of outputs from Pipeline objects

**Usage**

```r
generateOutput(object)
```

**Arguments**

- `object` object that contains input, pipeline, registry and output

**Details**

`generateOutput` is a generic function that is implemented for various types of pipeline objects such as `AnalysisPipeline` and `StreamingAnalysisPipeline`

The sequence of operations stored in the pipeline object are run and outputs generated, stored in a list

**Value**

Updated Pipeline object with the outputs at each step stored in the output slot.

Specific outputs can be obtained by using the `getOutputById` function
**generateReport**

Generate a HTML report from an AnalysisPipeline object

**Description**

Generate a HTML report from an AnalysisPipeline object

**Usage**

```r
generateReport(object, path)
```

`## S4 method for signature 'AnalysisPipeline,character'
generateReport(object, path = ".")`

**Arguments**

- `object` : object that contains input, pipeline, registry and output
- `path` : path on the file system, where the generated html report should be stored

**Details**

The sequence of operations stored in the AnalysisPipeline object are run, outputs generated, and a HTML report is generated with outputs in the same sequence as the pipeline created by the user

**Value**

Updated AnalysisPipeline object

**See Also**

Other Package core functions for batch/one-time analyses: `AnalysisPipeline-class, checkSchema, initialize, BaseAnalysisPipeline-method`
genericPipelineException

Examples

```r
## Not run:
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% univarCatDistPlots(uniCol = "Species", storeOutput = T) -> pipelineObj
pipelineObj %>>% generateReport(path = ".")

## End(Not run)
```

genericPipelineException

*Default exception for pipeline functions*

Description

Default exception for pipeline functions

Usage

```r
genericPipelineException(error)
```

Arguments

- `error` Error encountered during the execution of a particular pipeline function

Details

This function defines the default function which will be called in case of an exception occurring while executing any of the pipeline functions. While a function is registered, a custom function to deal with exceptions incurred during the call of the function being registered can be passed by the user. If passed, the custom function will be called instead of this function

See Also

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `createPipelineInstance`, `exportAsMetaPipeline`, `generateOutput`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPipeline`, `loadPredefinedFunctionRegistry`, `loadRegistry`, `prepExecution`, `registerFunction`, `savePipeline`, `saveRegistry`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`
getDatatype \hspace{1cm} Get Data Type

**Description**
Get Data Type

**Usage**
```python
getDatatype(dataset)
```

**Arguments**
- `dataset` a dataset which needs to be loaded

**Details**
Based on the datatype the columns are separated into categorical and numerical columns

**Value**
list with `numeric_cols` and `cat_cols`

**See Also**
Other Package EDA Utilities functions: `CheckColumnType, bivarPlots, correlationMatPlot, ignoreCols, multiVarOutlierPlot, outlierPlot, univarCatDistPlots`

**Examples**
```python
getDatatype(iris)
```

---

getFeaturesForPyClassification \hspace{1cm} Extracts selected columns from a data frame as a Python array

**Description**
Extracts selected columns from a data frame as a Python array

**Usage**
```python
getFeaturesForPyClassification(dataset, featureNames)
```
getInput

**Arguments**

- dataset: an R data frame
- featureNames: Column names to be extracted from the R data frames. A character vector.

**Details**

Helper function, which when provided an R data frame and a set of column/feature names, extracts them from the R data frame as a matrix and converts them to the equivalent Python array.

Typically this function can be used when providing a feature matrix to a Python machine learning function.

**See Also**

Other R helper utilities for Python: `getTargetForPyClassification`, `setPythonEnvir`

**Examples**

```r
## Not run:
getFeaturesForPyClassification(dataset = iris, 
featureNames = c("Sepal.Length", "Sepal.Width"))

## End(Not run)
```

getInput  

*Obtains the initializedInput*

**Description**

Obtains the initializedInput

**Usage**

```r
getInput(object)

## S4 method for signature 'BaseAnalysisPipeline'
getInput(object)
```

**Arguments**

- object: The AnalysisPipeline or StreamingAnalysisPipeline object

**Details**

Obtains the input from the AnalysisPipeline or StreamingAnalysisPipeline object

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class.
getValue

Value

Dataframe for an AnalysisPipeline & SparkDataFrame for a StreamingAnalysisPipeline

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getLoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize,BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% getInput

getLoggerDetails  Obtains the logger configuration for the pipeline

Description

Obtains the logger configuration for the pipeline

Usage

getLoggerDetails(object)

## S4 method for signature 'BaseAnalysisPipeline'
getLoggerDetails(object)

Arguments

object  A Pipeline object

Details

This function obtains the logger configuration for the pipeline.

Value

Logger configuration as a list
getOutputById

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

```r
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% getLoggerDetails
```

---

**getOutputById**  
 Obtains a specific output

**Description**

Obtains a specific output

**Usage**

```r
getOutputById(object, reqId, includeCall = F)
```

```r
## S4 method for signature 'BaseAnalysisPipeline'
getOutputById(object, reqId,
    includeCall = F)
```

**Arguments**

- `object`: The AnalysisPipeline or StreamingAnalysisPipeline object
- `reqId`: The position of the function for which the output is desired in the sequence of operations in the pipeline.
- `includeCall`: Logical which defines whether the call used to generate the output should be returned. By default this is false

**Details**

Obtains a specific output from the AnalysisPipeline or StreamingAnalysisPipeline object by passing the position of the function for which the output is desired, in the sequence of operations in the pipeline. This can be obtained by passing the number under the ‘id’ column in the pipeline table corresponding to the required function.

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class.
getPipeline

Obtain the pipeline

Description

Obtain the pipeline

Usage

getPipeline(object)

## S4 method for signature 'BaseAnalysisPipeline'
getPipeline(object)

Arguments

object The AnalysisPipeline or StreamingAnalysisPipeline object
getPipelinePrototype

Details

Obtains the pipeline from the AnalysisPipeline or StreamingAnalysisPipeline object as a tibble

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class

Value

Tibble describing the pipeline

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLogDetails, getOutputByld, getPipelinePrototype, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLogDetails, updateObject, visualizePipeline

Examples

```r
## Not run:
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
getNumRows <- function(dataset){
  return(nrow(dataset))
}
registerFunction("getNumRows")
pipelineObj %>% getNumRows %>% getPipeline

## End(Not run)
```
getRegistry

Arguments

metaPipelineObj

A MetaAnalysisPipeline object

Details

This method returns the prototype of functions in the pipeline and their respective arguments as proto object. Functions in the pipeline can be accessed easily by using the ‘$’ operator, and within the functions the arguments can be accessed the same way. These can be accessed and set to new values. This pipeline prototype can then be passed to the createPipelineInstance method which will instantiate an executable pipeline with the inputs set in the prototype

Value

An object of class proto from the 'proto' package

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getDf, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

## Not run:

pipelineObj <- AnalysisPipeline(input = iris)

pipelineObj %>% univarCatDistPlots(uniCol = "Species") %>%
exportAsMetaPipeline %>% getPipelinePrototype

## End(Not run)

---

getRegistry | Obtains the function registry

Description

Obtains the function registry

Usage

getRegistry()

Details

Obtains the function registry as a tibble, including both predefined and user defined functions
getResponse

Value

Tibble describing the registry

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLoggerDetails, getOutputById, getPipelinePrototype, getPipeline, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

getResponse()

### Description
Obtains the response term from the formula

### Usage
getResponse(f)

### Arguments

- **f**  
  formula from which term is to be extracted.

### Details
This is a helper function to extract the response variable from a formula

### Value
The response variable in the formula as a string

### Examples

```r
library(analysisPipelines)
getResponse(y ~ x1 + x2)
```
getTargetForPyClassification

Extracts selected column from a data frame a binary class Python array

Description

Extracts selected column from a data frame a binary class Python array

Usage

getTargetForPyClassification(dataset, targetVarName, positiveClass)

Arguments

dataset an R data frame
targetVarName Name of the target variable for classification. Should be a categorical variable.
positiveClass Name of the class of the target variable which should be coded as ’1’

Details

Helper function, which when provided an R dataframe and a binary categorical column, extracts it from the R data frame, converts it to 1/0 class coding, and converts it to a Python array

Typically this function can be used to extract a target variable for a classifier to be provided to a Python machine learning function

See Also

Other R helper utilities for Python: getFeaturesForPyClassification, setPythonEnvir

Examples

## Not run:
getTargetForPyClassification(dataset = iris,
targetVarName = "Species", positiveClass = "setosa")

## End(Not run)
**getTerm**  

*Obtains the dependency term from the formula*

**Description**

Obtains the dependency term from the formula

**Usage**

`getTerm(f)`

**Arguments**

- `f`  
  formula from which term is to be extracted.

**Details**

This is a helper function to extract the terms from a formula

**Value**

String with the terms

**Examples**

```r
library(analysisPipelines)
getTerm(y ~ x)
```

---

**ignoreCols**  

* Ignores the columns in the loaded dataframe object

**Description**

Ignores the columns in the loaded dataframe object

**Usage**

`ignoreCols(data, columns)`

**Arguments**

- `data`  
  the dataframe object that needs to be loaded
- `columns`  
  the names of columns to be ignored from dataframe object

**Details**

The columns selected are removed from the object
isDependencyParam

Value

Updated dataframe object

See Also

Other Package EDA Utilities functions: CheckColumnType, bivarPlots, correlationMatPlot, getDatatype, multiVarOutlierPlot, outlierPlot, univarCatDistPlots

Examples

ignoreCols(data = iris, columns = "Species")

library(analysisPipelines)

isDependencyParam(~f1)
loadMetaPipeline

Description

Load a meta-pipeline

Usage

loadMetaPipeline(path)

Arguments

path the path at which the .Rds file containing the pipeline is located

Details

This function loads a meta-pipeline from a file system, and returns the meta-pipeline object, which can be assigned to an object in the environment.

Note - When a meta-pipeline is loaded, the existing registry is overwritten with the registry saved with the meta-pipeline

Value

An MetaAnalysisPipeline object

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getILoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

```r
## Not run:
loadMetaPipeline(path = "./metaPipeline.RDS"

## End(Not run)
```
loadPipeline

**Description**

Loads the AnalysisPipeline or StreamingAnalysisPipeline object from the file system.

**Usage**

```r
loadPipeline(path, input = data.frame(), filePath = "")
```

**Arguments**

- `path` - the path at which the .Rds file containing the pipeline is located
- `input` - (optional) data frame with which the pipeline object should be initialized
- `filePath` - (optional) path where a dataset in .CSV format is present which is to be loaded

**Details**

The AnalysisPipeline or StreamingAnalysisPipeline object is loaded into the file system from the file system based on the path specified.

Optionally, the `input` parameter can be provided to initialize the AnalysisPipeline or StreamingAnalysisPipeline object with an R data frame or Streaming Spark DataFrame (in case of StreamingAnalysisPipeline object) present in the R session.

Another provided option, is to specify a `filePath` where the input dataset is present (in a .CSV format) and the object will be initialized with this data frame. The `filePath` parameter takes precedence over `input` parameter. This is applicable only from AnalysisPipeline objects.

**Note** - When a pipeline is loaded, the existing registry is overwritten with the registry saved with the pipeline.

**Value**

An AnalysisPipeline or StreamingAnalysisPipeline object, optionally initialized with the data frame provided.

**See Also**

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `createPipelineInstance`, `exportAsMetaPipeline`, `generateOutput`, `genericPipelineException`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPredefinedFunctionRegistry`, `loadRegistry`, `prepExecution`, `registerFunction`, `savePipeline`, `saveRegistry`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`
Examples

```r
## Not run:
library(analysisPipelines)
loadPipeline(path = "/pipeline.RDS"

## End(Not run)
```

loadPredefinedFunctionRegistry

Loading the registry of predefined functions

Description

Loading the registry of predefined functions

Usage

```r
loadPredefinedFunctionRegistry()
```

Details

Loads the registry of predefined functions

See Also

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `createPipelineInstance`, `exportAsMetaPipeline`, `generateOutput`, `genericPipelineException`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPipeline`, `loadRegistry`, `prepExecution`, `registerFunction`, `savePipeline`, `saveRegistry`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`

Examples

```r
## Not run:
library(analysisPipelines)
loadPredefinedFunctionRegistry()

## End(Not run)
```
loadRegistry  
*Loads a function registry from a file*

**Description**

Loads a function registry from a file

**Usage**

```r
loadRegistry(path)
```

**Arguments**

- `path` path on the file system, where the registry is to be loaded from

**Details**

This function loads a function registry and associated function definition stored in an RDS file into the environment. The existing registry is overwritten with the newly loaded registry

**See Also**

Other Package core functions: `BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLogDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline`

**Examples**

```r
## Not run:
library(analysisPipelines)
loadRegistry(path = "./registry.RDS")

## End(Not run)
```

---

**MetaAnalysisPipeline-class**

*Class for creating and working with meta-pipelines*

**Description**

Class for creating and working with meta-pipelines
multiVarOutlierPlot

Details

This class works with the AnalysisPipeline and StreamingAnalysisPipeline classes, and allows the pipeline to be exported as meta-pipeline. A meta-pipeline is a construct, where the input dataset as well as the arguments to functions in the pipeline are not defined. Only the analysis flow and dependencies are stored.

Slots

pipeline  A tibble which holds functions to be called in the pipeline
pipelinePrototype  An object of class proto from the 'proto' package which maintains the prototype of the functions in the pipeline and their respective arguments
type  A string defining whether it is a batch or streaming pipeline. Acceptable values are 'batch' & 'streaming'

See Also

Other Package core functions: BaseAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInputModule, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize,BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

multiVarOutlierPlot  Multi-Variate Outlier Plot

Description

Multi-Variate Outlier Plot

Usage

multiVarOutlierPlot(data, depCol, indepCol, sizeCol, priColor = "blue", optionalPlots = 0, cutoffValue = 0.05)

Arguments

data  the dataframe that needs to be loaded
depCol  the name of column which is to be identified as dependent column
indepCol  the name of an independent column
sizeCol  the name of column used to define the size of point in plots
priColor  the primary color for the plots
optionalPlots  A Flag for optional plots
cutoffValue  A p-value cutoff for detecting outliers
Details

Multivariate outlier plot using the selected columns from the dataframe

Value

Outliers plot

See Also

Other Package EDA Utilities functions: CheckColumnType, bivarPlots, correlationMatPlot, getDatatype, ignoreCols, outlierPlot, univarCatDistPlots

Examples

```r
## Not run:
multiVarOutlierPlot(data = iris, depCol = "Sepal.Length",
                    indepCol = "Sepal.Width", sizeCol = "Petal.Length")
## End(Not run)
```

---

**outlierPlot**  
*Outlier detection plot*

Description

Outlier detection plot

Usage

```r
outlierPlot(data, method = "iqr", columnName, cutoffValue = 0.05,
            priColor = "blue", optionalPlots = 0)
```

Arguments

- **data**: the dataframe that needs to be loaded
- **method**: the method on which outliers are to be identified
- **columnName**: the name of column for which the outliers are identified
- **cutoffValue**: the cut off value to define the threshold for outliers
- **priColor**: the primary color for the plots
- **optionalPlots**: A Flag for optional plots

Details

Outlier are to be identified on the selected column from the dataframe
prepExecution

Value
Outliers plot object

See Also
Other Package EDA Utilities functions: CheckColumnType, bivarPlots, correlationMatPlot, getDatatype, ignoreCols, multiVarOutlierPlot, univarCatDistPlots

Examples
## Not run:
outlierPlot(data = iris, columnName = "Sepal.Length")
## End(Not run)

prepExecution Prepare the pipeline for execution

Description
Prepare the pipeline for execution

Usage
prepExecution(object)

## S4 method for signature 'BaseAnalysisPipeline'
prepExecution(object)

Arguments
object A Pipeline object

Details
The pipeline is prepared for execution by identifying the graph of the pipeline as well as its topological ordering, and dependency map in order to prepare for execution

Value
Updated AnalysisPipeline StreamingAnalysisPipeline object

See Also
Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getILoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDFBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline
Examples

## Not run:
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% univarCatDistPlots(uniCol = "Species", priColor = "blue", optionalPlots = 0, storeOutput = T) %>>% prepExecution -> pipelineObj

## End(Not run)

registerFunction

Register a user-defined function to be used with a AnalysisPipeline or StreamingAnalysisPipeline object

Description

Register a user-defined function to be used with a AnalysisPipeline or StreamingAnalysisPipeline object

Usage

registerFunction(functionName, heading = "", functionType = "batch", engine = "r", exceptionFunction = as.character(substitute(genericPipelineException)), isDataFunction = T, firstArgClass = "", loadPipeline = F, userDefined = T)

Arguments

functionName: name of function to be registered
heading: heading of that section in report
functionType: type of function - 'batch' for AnalysisPipeline objects, 'streaming' for StreamingAnalysisPipeline objects
engine: specifies which engine the function is to be run on. Available engines include "r", "spark", and "python"
exceptionFunction: R object corresponding to the exception function
isDataFunction: logical parameter which defines whether the function to be registered operates on data i.e. the first parameter is a dataframe
firstArgClass: character string with the class of the first argument to the function, if it is a non-data function
loadPipeline: logical parameter to see if function is being used in loadPipeline or not. This is for internal working
userDefined: logical parameter defining whether the function is user defined. By default, set to true
savePipeline

Details

The specified operation along with the heading and engine details is stored in the registry, after which it can be added to a pipeline.

If the function already exists in the registry, registration will be skipped. In order to change the definition, the function needs to be reassigned in the Global Environment and then the `registerFunction` called again.

See Also

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `createPipelineInstance`, `exportAsMetaPipeline`, `generateOutput`, `genericPipelineException`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPipeline`, `loadPredefinedFunctionRegistry`, `loadRegistry`, `prepExecution`, `savePipeline`, `saveRegistry`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`

Examples

```r
## Not run:
library(analysisPipelines)
getNumRows <- function(dataset){
  return(nrow(dataset))
}

registerFunction("getNumRows")

## End(Not run)
```

savePipeline

`Saves the AnalysisPipeline or StreamingAnalysisPipeline object to the file system without outputs`

Description

Saves the AnalysisPipeline or StreamingAnalysisPipeline object to the file system without outputs

Usage

```r
savePipeline(object, path)

## S4 method for signature 'BaseAnalysisPipeline'
savePipeline(object, path)

## S4 method for signature 'MetaAnalysisPipeline'
savePipeline(object, path)
```
saveRegistry

Arguments

- **object**: object that contains input, pipeline, registry and output
- **path**: the path at which the .Rda file containing the pipeline should be stored, along
  with the name of the file including a .Rda extension

Details

The AnalysisPipeline or StreamingAnalysisPipeline object is saved to the file system in the
paths specified

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class

Value

Does not return a value

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getILoggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, saveRegistry, setInput, setLoggerDetails, updateObject, visualizePipeline

Examples

```r
## Not run:
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% savePipeline(path = ".\test.RDS")

## End(Not run)
```

---

**Description**

Saves the registry to the file system

**Usage**

```
saveRegistry(path)
```

**Arguments**

- **path**: path on the file system, where the registry is to be saved to
**setInput**

37

**Details**

This function saves the existing function registry and associated function definition loaded in the environment into a file.

**See Also**

Other Package core functions: `BaseAnalysisPipeline-class`, `MetaAnalysisPipeline-class`, `assessEngineSetUp`, `checkSchemaMatch`, `createPipelineInstance`, `exportAsMetaPipeline`, `generateOutput`, `genericPipelineException`, `getInput`, `getLoggerDetails`, `getOutputById`, `getPipelinePrototype`, `getPipeline`, `getRegistry`, `initDfBasedOnType`, `initialize`, `BaseAnalysisPipeline-method`, `loadMetaPipeline`, `loadPipeline`, `loadPredefinedFunctionRegistry`, `loadRegistry`, `prepExecution`, `registerFunction`, `savePipeline`, `setInput`, `setLoggerDetails`, `updateObject`, `visualizePipeline`

**Examples**

```r
## Not run:
library(analysisPipelines)
saveRegistry(path = "./registry.RDS")

## End(Not run)
```

---

**setInput**

Sets the input for an AnalysisPipeline or StreamingAnalysisPipeline object

**Description**

Sets the input for an AnalysisPipeline or StreamingAnalysisPipeline object

**Usage**

```r
setInput(object, input, filePath = "")
```

```r
## S4 method for signature 'BaseAnalysisPipeline'
setInput(object, input, filePath = "")
```

**Arguments**

- `object`:
  - object that contains input, pipeline, registry and output
- `input`:
  - the input data frame
- `filePath`:
  - path to the file which needs to be read (currently supports .csv files)

**Details**

Assigns the input to the pipeline for an AnalysisPipeline or StreamingAnalysisPipeline object

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class
setLoggerDetails

Value

Updated AnalysisPipeline StreamingAnalysisPipeline object

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getLogDetails, getOutputByld, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setLoggerDetails, updateObject, visualizePipeline

Examples

library(analysisPipelines)
pipelineObj <- AnalysisPipeline()
pipelineObj %>>% setInput(input = iris) -> pipelineObj

Description

Sets the logger configuration for the pipeline

Usage

setLoggerDetails(object, target = "console",
                targetFile = "pipelineExecution.out", layout = "layout.simple")

## S4 method for signature 'BaseAnalysisPipeline'
setLoggerDetails(object,
                 target = "console", targetFile = "pipelineExecution.out",
                 layout = "layout.simple")

Arguments

- **object**: A Pipeline object
- **target**: A string value. 'console' for appending to console, 'file' for appending to a file, or 'console&file' for both
- **targetFile**: File name of the log file in case the target is 'file'
- **layout**: Specify the layout according to 'futile.logger' package convention

Details

This function sets the logger configuration for the pipeline.
setPythonEnvir

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getPythonDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, updateObject, visualizePipeline

Examples

library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>>% setLoggerDetails(target = "file",
  targetFile = "pipeline.out") -> pipelineObj

---

setPythonEnvir  Sets the python environment to be used

Description

Sets the python environment to be used

Usage

setPythonEnvir(type = "conda", pathOrEnvirName = "base")

Arguments

type  Type of python environment. Takes three possible values - 'conda' for Anaconda environments, 'virtualenv' for Virtual environments, and 'python' to manually set the python path to use

pathOrEnvirName  Name of the environment for Anaconda and Virtual environments, or the Python path when type is 'python'

Details

Wrapper function over reticulate functions to set a python environment to be used

See Also

Other R helper utilities for Python: getFeaturesForPyClassification, getTargetForPyClassification

Examples

## Not run:
setPythonEnvir()

## End(Not run)
sparkRSessionCreateIfNotPresent

Connect to a Spark session

Description

Connect to a Spark session

Usage

sparkRSessionCreateIfNotPresent(...)

Arguments

... Arguments to sparkR.session

Details

Loads the SparkR package and initializes a Spark session from R

See Also

Other Spark utilities: castKafkaStreamAsString, convertKafkaValueFromJson

Examples

```r
## Not run:
sparkHome <- "/Users/naren/softwares/spark-2.3.1-bin-hadoop2.7/
sparkMaster <- "local[1]"
sparkPackages <- c("org.apache.spark:spark-sql-kafka-0-10_2.11:2.3.1")
sparkRSessionCreateIfNotPresent(master = sparkMaster,
sparkPackages = sparkPackages)
## End(Not run)
```

StreamingAnalysisPipeline-class

Class for constructing Analysis Pipelines for streaming analyses

Description

Class for constructing Analysis Pipelines for streaming analyses
Details

Inherits the base class `BaseAnalysisPipeline` class which holds the metadata including the registry of available functions, the data on which the pipeline is to be applied, as well as the pipeline itself.

This class currently only supports Apache Spark Structured Streaming, implemented through the SparkR interface.

Slots

- `input`: The input Spark DataFrame on which analysis is to be performed.
- `originalSchemaDf`: Empty Spark DataFrame representing the schema of the input.

### Description

**Univariate Categoric Distribution**

### Usage

```r
univarCatDistPlots(data, uniCol, priColor = "blue", optionalPlots = 0)
```

### Arguments

- `data`: the dataset where the column on which the plot is to be generated is present.
- `uniCol`: the name of column on which the plot needs to be generated.
- `priColor`: the primary color for the plots.
- `optionalPlots`: A Flag for optional plots.

### Details

A univariate distribution graph on the selected categorical columns from the dataframe.

### Value

A univariate categoric distribution plot.

### See Also

Other Package EDA Utilities functions: `CheckColumnType`, `bivarPlots`, `correlationMatPlot`, `getDatatype`, `ignoreCols`, `multiVarOutlierPlot`, `outlierPlot`.

### Examples

```r
univarCatDistPlots(data = iris, uniCol = "Species")
```
updateObject

Update the AnalysisPipeline or StreamingAnalysisPipeline object by adding an operation to the pipeline

Description

Update the AnalysisPipeline or StreamingAnalysisPipeline object by adding an operation to the pipeline

Usage

```
updateObject(object, operation, heading = "", parameters, outAsIn = F, storeOutput = F)
```

## S4 method for signature 'BaseAnalysisPipeline'

```
updateObject(object, operation,
             heading = "", parameters, outAsIn = F, storeOutput = F)
```

Arguments

- `object`: object that contains input, pipeline, registry and output
- `operation`: function name to be updated in tibble
- `heading`: heading of that section in report
- `parameters`: parameters passed to that function
- `outAsIn`: whether to use original input or output from previous function
- `storeOutput`: whether the output of this operation is to be stored

Details

The specified operation along with the heading and parameters is updated in the pipeline slot of the AnalysisPipeline or StreamingAnalysisPipeline object, where the sequence of operations to be performed is stored.

This method is implemented on the base class as it is a shared functionality types of Analysis Pipelines which extend this class

Value

Updated AnalysisPipeline StreamingAnalysisPipeline object

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEnginesetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, loggerDetails, getOutputById, getPipelinePrototype, getPipeline, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, visualizePipeline
visualizePipeline

Visualizes the pipeline as a graph

Description

Visualizes the pipeline as a graph

Usage

visualizePipeline(object)

Arguments

object

The AnalysisPipeline or StreamingAnalysisPipeline object

Details

Indicates dependencies amongst functions as well as functions for which output needs to be stored

Value

A graph object which can be printed (or) plotted to visualize the pipeline

See Also

Other Package core functions: BaseAnalysisPipeline-class, MetaAnalysisPipeline-class, assessEngineSetUp, checkSchemaMatch, createPipelineInstance, exportAsMetaPipeline, generateOutput, genericPipelineException, getInput, getPipeline, getOutputById, getPipelinePrototype, getRegistry, initDfBasedOnType, initialize, BaseAnalysisPipeline-method, loadMetaPipeline, loadPipeline, loadPredefinedFunctionRegistry, loadRegistry, prepExecution, registerFunction, savePipeline, saveRegistry, setInput, setLoggerDetails, updateObject

Examples

## Not run:
library(analysisPipelines)
pipelineObj <- AnalysisPipeline(input = iris)
pipelineObj %>% univarCatDistPlots(uniCol = "Species", priColor = "blue", optionalPlots = 0, storeOutput = T) %>% visualizePipeline

## End(Not run)
Index

AnalysisPipeline (AnalysisPipeline-class) 3
AnalysisPipeline-class 3
analysisPipelines 4
analysisPipelines-package (analysisPipelines) 4
assessEngineSetUp, 4, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
assessEngineSetUp, BaseAnalysisPipeline-method (assessEngineSetUp) 4
BaseAnalysisPipeline 3, 41
BaseAnalysisPipeline (BaseAnalysisPipeline-class) 5
BaseAnalysisPipeline-class 5
bivarPlots, 6, 8, 10, 16, 26, 32, 33, 41
castKafkaValueFromJson, 7, 9, 40
CheckColumnType 4, 14
checkSchemaMatch, 5, 6, 8, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
checkSchemaMatch, AnalysisPipeline-method (checkSchemaMatch) 8
correlationMatPlot, 7, 8, 10, 16, 26, 32, 33, 41
createPipelineInstance, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
createPipelineInstance, MetaAnalysisPipeline-method (createPipelineInstance) 11
datatypes, 21, 22, 23, 27–31, 33, 35–39, 42, 43
datatypes, PipelinePrototype-method (datatypes) 21
generateOutput, 5, 6, 9, 11, 12, 13, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
generateOutput, AnalysisPipeline-method (generateOutput) 13
getDatatype, 7, 8, 10, 16, 26, 32, 33, 41
getFeaturesForPyClassification, 16, 24, 39
genericPipelineException, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
generateReport, 4, 14
generateReport, AnalysisPipeline, character-method (generateReport) 14
generateReport, AnalysisPipeline-method (generateReport) 14
getInput, 5, 6, 9, 11, 12, 14, 15, 17, 19–23, 27–31, 33, 35–39, 42, 43
generateReport, MetaAnalysisPipeline-method (generateReport) 14
getInput, BaseAnalysisPipeline-method (getInput) 17
getLoggerDetails, 5, 6, 9, 11, 12, 14, 15, 18, 18, 20–23, 27–31, 33, 35–39, 42, 43
generateReport, MetaAnalysisPipeline-method (getLoggerDetails) 18
generateReport, MetaAnalysisPipeline-method (getLoggerDetails) 18
generateOutput, StreamingAnalysisPipeline-method (generateOutput) 13
getInputById, 5, 6, 9, 11–15, 18, 19, 21–23, 27–31, 33, 35–39, 42, 43
generateOutput, MetaAnalysisPipeline-method (getInputById) 19
generateOutput, MetaAnalysisPipeline-method (getInputById) 19
generateOutput, MetaAnalysisPipeline-method (getInputById) 19
goPipeLine, 5, 6, 9, 11, 12, 14, 15, 18–20, 20, 22, 23, 27–31, 33, 35–39, 42, 43
generateOutput, MetaAnalysisPipeline-method (goPipeLine) 20
getPipelinePrototype, 5, 6, 9, 11, 12, 14, 15, 18–22, 22, 27–31, 33, 35–39, 42, 43
generateOutput, MetaAnalysisPipeline-method (getPipelinePrototype) 21
generateOutput, MetaAnalysisPipeline-method (getPipelinePrototype) 21
getRegistry, 5, 6, 9, 11, 12, 14, 15, 18–22, 22, 27–31, 33, 35–39, 42, 43
generateOutput, MetaAnalysisPipeline-method (getRegistry) 23
generateOutput, MetaAnalysisPipeline-method (getRegistry) 23
getTargetForPyClassification, 17, 24, 39
generateOutput, MetaAnalysisPipeline-method (getTargetForPyClassification) 24
generateOutput, MetaAnalysisPipeline-method (getTargetForPyClassification) 24
getTerm, 25
ignoreCols, 7, 8, 10, 16, 25, 32, 33, 41

44
initDfBasedOnType, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
isDependencyParam, 26
loadMetaPipeline, 5, 6, 9, 11, 12, 14, 15, 18–23, 27, 28–31, 33, 35–39, 42, 43
loadPipeline, 5, 6, 9, 11, 12, 14, 15, 18–23, 27, 28, 29–31, 33, 35–39, 42, 43
loadPredefinedFunctionRegistry, 5, 6, 9, 11, 12, 14, 15, 18–23, 27, 28, 29, 30, 31, 33, 35–39, 42, 43
loadRegistry, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–29, 30, 31, 33, 35–39, 42, 43
MetaAnalysisPipeline (MetaAnalysisPipeline-class), 30
MetaAnalysisPipeline-class, 30
multiVarOutlierPlot, 7, 8, 10, 16, 26, 31, 33, 41
outlierPlot, 7, 8, 10, 16, 26, 32, 32, 41
prepExecution, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
prepExecution, BaseAnalysisPipeline-method (prepExecution), 33
registerFunction, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 34, 36–39, 42, 43
savePipeline, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35, 35–39, 42, 43
savePipeline, BaseAnalysisPipeline-method (savePipeline), 35
savePipeline, MetaAnalysisPipeline-method (savePipeline), 35
saveRegistry, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35, 36, 36, 38, 39, 42, 43
setInput, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–37, 37, 39, 42, 43
setInput, BaseAnalysisPipeline-method (setInput), 37
setLoggerDetails, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–38, 38, 42, 43
setLoggerDetails, BaseAnalysisPipeline-method (setLoggerDetails), 38
setPythonEnvir, 17, 24, 39
sparkRSessionCreateIfNotPresent, 7, 10, 40
StreamingAnalysisPipeline (StreamingAnalysisPipeline-class), 40
StreamingAnalysisPipeline-class, 40
univarCatDistPlots, 7, 8, 10, 16, 26, 32, 33, 41
updateObject, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
updateObject, BaseAnalysisPipeline-method (updateObject), 42
visualizePipeline, 5, 6, 9, 11, 12, 14, 15, 18–23, 27–31, 33, 35–39, 42, 43
visualizePipeline, BaseAnalysisPipeline-method (visualizePipeline), 43
visualizePipeline, MetaAnalysisPipeline-method (visualizePipeline), 43