Eskapade-Spark Documentation

KPMG Advanced Analytics
Big Data team

Feb 23, 2019

Contents

1 Release notes **3**
- 1.1 Version 1.0 . . . 3
- 1.2 Version 0.9 . . . 3

2 Installation **5**
- 2.1 requirements . . . 5
- 2.2 pypi . . . 5
- 2.3 github . . . 5
- 2.4 python . . . 5

3 Quick run **7**

4 Contact and support **9**

5 Contents **11**
- 5.1 Tutorials . . . 11
- 5.2 Release notes . . . 17
- 5.3 Developing and Contributing . . . 17
- 5.4 References . . . 18
- 5.5 API . . . 18
- 5.6 Appendices . . . 52
- 5.7 Indices and tables . . . 55

Python Module Index **57**

- Version: 1.0.0
- Released: Feb 2019

Eskapade is a light-weight, python-based data analysis framework, meant for modularizing all sorts of data analysis problems into reusable analysis components. For documentation on Eskapade, please go to this link.

Eskapade-Spark is the Spark-based extension of Eskapade. For documentation on Eskapade-Spark, please go here.

CHAPTER 1

Release notes

1.1 Version 1.0

Eskapade-Spark v1.0 (February 2019) is in synch with Eskapade-Core v1.0 and Eskapade v1.0, contains several small upgrades wrt v0.9:

- Minor upgrades to spark_histogrammar_filler link.
- Include hive_reader and hive_writer links, for working with hive tables.
- Include jdbc module, for opening a connection to a jdbc database, and a jdbc_reader link.

1.2 Version 0.9

Eskapade-Spark v0.9 (December 2018) contains only one update compared with v0.8:

- All code has been updated to Eskapade v0.9, where the core functionality has been split off into the Eskapade-Core package. As such the code is backwards-incompatible with v0.8.

See release notes for previous versions of Eskapade-Spark.

CHAPTER 2

Installation

2.1 requirements

Eskapade-Spark requires `Python 3.5+`, `Eskapade v0.8+` and `Spark v2.1.2`. These are pre-installed in the Eskapade docker.

2.2 pypi

To install the package from pypi, do:

```
$ pip install Eskapade-Spark
```

2.3 github

Alternatively, you can check out the repository from github and install it yourself:

```
$ git clone https://github.com/KaveIO/Eskapade-Spark.git eskapade-spark
```

To (re)install the python code from your local directory, type from the top directory:

```
$ pip install -e eskapade-spark
```

2.4 python

After installation, you can now do in Python:

```
import eskapadespark
```

Congratulations, you are now ready to use Eskapade-Spark!

CHAPTER 3

Quick run

To see the available Eskapade-Spark examples, do:

```
$ export TUTDIR=`pip show Eskapade-Spark | grep Location | awk '{ print $2"/
↪eskapadespark/tutorials" }'`
$ ls -l $TUTDIR/
```

E.g. you can now run:

```
$ eskapade_run $TUTDIR/esk601_spark_configuration.py
```

For all available examples, please see the tutorials.

CHAPTER 4

Contact and support

Contact us at: kave [at] kpmg [dot] com

Please note that the KPMG Eskapade group provides support only on a best-effort basis.

CHAPTER 5

Contents

5.1 Tutorials

This section contains materials on how to use Eskapade-Spark. All command examples can be run from any directory with write access. For more in depth explanations on the functionality of the code-base, try the API docs.

5.1.1 All Spark Examples in Eskapade

All Eskapade-Spark example macros can be found in the tutorials directory. For ease of use, let's make a shortcut to the directory containing the tutorials:

```
$ export TUTDIR=`pip show Eskapade-Spark | grep Location | awk '{ print $2"/
↪eskapadespark/tutorials" }'`
$ ls -l $TUTDIR/
```

The numbering of the example macros follows the package structure:

- `esk600+`: macros for processing Spark datasets and performing analysis with Spark.

These macros are briefly described below. You are encouraged to run all examples to see what they can do for you!

Example esk601: setting the spark configuration

Tutorial macro for configuring Spark in multiple ways.

```
$ eskapade_run $TUTDIR/esk601_spark_configuration.py
```

Example esk602: reading csv to a spark dataframe

Tutorial macro for reading CSV files into a Spark data frame.

```
$ eskapade_run $TUTDIR/esk602_read_csv_to_spark_df.py
```

Example esk603: writing spark data to csv

Tutorial macro for writing Spark data to a CSV file.

```
$ eskapade_run $TUTDIR/esk603_write_spark_data_to_csv.py
```

Example esk604: executing queries

Tutorial macro for applying a SQL-query to one more objects in the DataStore. Such SQL-queries can for instance be used to filter data.

```
$ eskapade_run $TUTDIR/esk604_spark_execute_query.py
```

Example esk605: creating Spark data frames from various input data

Tutorial macro for creating Spark data frames from different types of input data.

```
$ eskapade_run $TUTDIR/esk605_create_spark_df.py
```

Example esk606: converting Spark data frames into different data types

Tutorial macro for converting Spark data frames into a different data type and apply transformation functions on the resulting data.

```
$ eskapade_run $TUTDIR/esk606_convert_spark_df.py
```

Example esk607: adding a new column to a Spark dataframe

Tutorial macro for adding a new column to a Spark dataframe by applying a Spark built-in or user-defined function to a selection of columns in a Spark dataframe.

```
$ eskapade_run $TUTDIR/esk607_spark_with_column.py
```

Example esk608: making histograms of a Spark dataframe

Tutorial macro for making histograms of a Spark dataframe using the Histogrammar package.

```
$ eskapade_run $TUTDIR/esk608_spark_histogrammar.py
```

Example esk609: applying map functions on groups of rows

Tutorial macro for applying map functions on groups of rows in Spark data frames.

```
$ eskapade_run $TUTDIR/esk609_map_df_groups.py
```

Example esk610: running Spark Streaming word count example

Tutorial macro running Spark Streaming word count example in Eskapade, derived from:

https://spark.apache.org/docs/latest/streaming-programming-guide.html

Counts words in UTF8 encoded, 'n' delimited text received from a stream every second. The stream can be from either files or network.

```
$ eskapade_run $TUTDIR/esk610_spark_streaming_wordcount.py
```

Example esk611: techniques for flattening a time-series in Spark

This macro demonstrates techniques for flattening a time-series in Spark.

```
$ eskapade_run $TUTDIR/esk611_flatten_time_series.py
```

5.1.2 Tutorial 6: going Spark

This section provides a tutorial on how to use Apache Spark in Eskapade. Spark works 'out of the box' in the Eskapade docker/vagrant image. For details on how to setup a custom Spark setup, see the Spark section in the Appendix.

In this tutorial we will basically redo Tutorial 1 but use Spark instead of Pandas for data processing. The following paragraphs describe step-by-step how to run a Spark job, use existing links and write your own links for Spark queries.

Note: To get familiar with Spark in Eskapade you can follow the exercises in `python/eskapadespark/tutorials/tutorial_6.py`.

Running the tutorial macro

The very first step to run the tutorial Spark job is:

```
$ eskapade_run python/eskapadespark/tutorials/tutorial_6.py
```

Eskapade will start a Spark session, do nothing, and quit - there are no chains/links defined yet. The Spark session is created via the `SparkManager` which, like the `DataStore`, is a singleton that configures and controls Spark sessions centrally. It is activated through the magic line:

```
process_manager.service(SparkManager).create_session(include_eskapade_modules=True)
```

Note that when the Spark session is created, the following line appears in logs:

```
Adding Python modules to egg archive <PATH_TO_ESKAPADE>/lib/es_python_modules.egg
```

This is the `SparkManager` that ensures all Eskapade source code is uploaded and available to the Spark cluster when running in a distributed environment. To include the Eskapade code the argument `include_eskapade_modules` need to be set to `True` (by default it is `False`).

If there was an `ImportError: No module named pyspark` then, most likely, `SPARK_HOME` and `PYTHONPATH` are not set up correctly. For details, see the Spark section in the Appendix.

Reading data

Spark can read data from various sources, e.g. local disk, HDFS, HIVE tables. Eskapade provides the `SparkDfReader` link that uses the `pyspark.sql.DataFrameReader` to read flat CSV files into Spark DataFrames, RDD's, and Pandas DataFrames. To read in the Tutorial data, the following link should be added to the `Data` chain:

```
data = Chain('Data')
reader = SparkDfReader(name='Read_LA_ozone', store_key='data', read_methods=['csv'])
reader.read_meth_args['csv'] = (DATA_FILE_PATH,)
reader.read_meth_kwargs['csv'] = dict(sep=',', header=True, inferSchema=True)
data.add(reader)
```

The `DataStore` holds a pointer to the Spark dataframe in (distributed) memory. This is different from a Pandas dataframe, where the entire dataframe is stored in the `DataStore`, because a Spark dataframe residing on the cluster may not fit entirely in the memory of the machine running Eskapade. This means that Spark dataframes are never written to disk in `DataStore` pickles!

Using existing links

Spark has a large set of standard functions for Spark DataFrames and RDD's. Although the purpose of Eskapade is not to duplicate this functionality, there are some links created for generic functions to facilitate specifying Spark queries directly in the macro, instead of hard-coding them in links. This is handy for bookkeeping queries at a central place and reducing code duplication, especially for smaller analysis steps. For example, the `SparkExecuteQuery` link takes any string containig SQL statements to perform a custom query with Spark on a dataframe.

Column transformations

To add two columns to the Tutorial data using the conversion functions defined earlier in the macro, two `SparkWithColumn` links need to be added to the `Data` chain, one for each additional column:

```
from pyspark.sql.functions import udf
from pyspark.sql.types import TimestampType, FloatType
...

transform = SparkWithColumn(name='Transform_doy', read_key=reader.store_key,
                            store_key='transformed_data', col_select=['doy'],
                            func=udf(comp_date, TimestampType()), new_column='date')
data.add(transform)
transform = SparkWithColumn(name='Transform_vis', read_key=transform.store_key,
                            store_key='transformed_data', col_select=['vis'],
                            func=udf(mi_to_km, FloatType()), new_column='vis_km')
data.add(transform)
```

Note that the functions defined in the macro are converted to user-defined functions with `pyspark.sql.functions.udf` and their output types are explicitly specified in terms of `pyspark.sql.types`. Omitting these type definitions can lead to obscure errors when executing the job.

Creating custom links

More complex queries deserve their own links since links provide full flexibility w.r.t. specifying custom data operation. For this Tutorial the 'complex query' is to just print 42 rows of the Spark dataframe. Of course, more advanced Spark functions can be applied in a similar fashion. A link is created just like was done before, e.g.:

```
$ eskapade_generate_link --dir python/eskapadespark/links SparkDfPrinter
```

This creates the link `python/eskapadespark/links/sparkdfprinter.py`. Do not forget to include the `import` statements in the `__init__.py` file as indicated by the `eskapade_generate_link` command.

The next step is to add the desired functionality to the link. In this case, the Spark dataframe needs to be retrieved from the `DataStore` and a `show()` method of that dataframe needs to be executed. The `execute()` method of the link is the right location for this:

```
def execute(self):
  """Execute the link.

  :returns: status code of execution
  :rtype: StatusCode
  """
  settings = process_manager.service(ConfigObject)
  ds = process_manager.service(DataStore)

  # --- your algorithm code goes here
  self.logger.debug('Now executing link: {link}.', link=self.name)
  df = ds[self.read_key]
  df.show(self.nrows)

  return StatusCode.Success
```

There is an additional attribute `self.nrows` which should be set in the link. By default, a generated link process only the `read_key` and `store_key` arguments and fails if there are any residual kwargs. To set the `nrows` attribute, add `nrows` to the key-value arguments in the `__init__()` method:

```
def __init__(self, **kwargs):
    ...

    self._process_kwargs(kwargs, read_key=None, store_key=None, nrows=1)
```

In order to configure Eskapade to run this link, the link needs to be added to a chain, e.g. `Summary`, in the `tutorial/tutorial_6.py` macro. This should look similar to:

```
printer = SparkDfPrinter(name='Print_spark_df', read_key=transform.store_key,␣
↪nrows=42)
summary.add(printer)
```

The name of the dataframe is the output name of the `transform` link and the number of rows to print is specified by the `nrows` parameter.

Eskapade should now be ready to finally execute the macro and provide the desired output:

```
$ eskapade_run python/eskapadespark/tutorials/tutorial_6.py

* * * Welcome to Eskapade * * *
...

+-----+----+----+--------+----+----+---+---+---+---+--------------------+--------+
|ozone|  vh|wind|humidity|temp| ibh|dpg|ibt|vis|doy|                date|  vis_km|
+-----+----+----+--------+----+----+---+---+---+---+--------------------+--------+
|    3|5710|   4|      28|  40|2693|-25| 87|250|  3|1976-01-03 00:00:...| 402.335|
|    5|5700|   3|      37|  45| 590|-24|128|100|  4|1976-01-04 00:00:...| 160.934|
|    5|5760|   3|      51|  54|1450| 25|139| 60|  5|1976-01-05 00:00:...| 96.5604|
```

(continues on next page)

(continued from previous page)

```
...

|    6|5700|   4|      86|  55|2398| 21|121|200| 44|1976-02-13 00:00:...| 321.868|
|    4|5650|   5|      61|  41|5000| 51| 24|100| 45|1976-02-14 00:00:...| 160.934|
|    3|5610|   5|      62|  41|4281| 42| 52|250| 46|1976-02-15 00:00:...| 402.335|
+-----+----+----+--------+----+----+---+---+---+---+--------------------+--------+
only showing top 42 rows
...

* * * Leaving Eskapade. Bye! * * *
```

That's it!

Spark Streaming

Eskapade supports the use of Spark Streaming as demonstrated in the word count example `tutorials/esk610_spark_streaming_wordcount.py`. The data is processed in (near) real-time as micro batches of RDD's, so-called discretized streaming, where the stream originates from either new incoming files or network connection. As with regular Spark queries, various transformations can be defined and applied in subsequent Eskapade links.

For details on Spark Streaming, see also https://spark.apache.org/docs/2.1.1/streaming-programming-guide.html.

File stream

The word count example using the file stream method can be run by executing in two different terminals:

```
terminal 1 $ eskapade_run -c stream_type='file' python/eskapadespark/tutorials/esk610_
↪spark_streaming_wordcount.py

terminal 2 $ mkdir /tmp/eskapade_stream_test/
terminal 2 $ for ((i=0; i<=100; i++)); do echo "Hello world" > /tmp/eskapade_stream_
↪test/dummy_$(printf %05d ${i}); sleep 0.2; done
```

Where bash `for`-loop will create a new file containing `Hello world` in the /tmp/eskapade_stream_test directory every 0.2 second. Spark Streaming will pick up and process these files and in `terminal 1` a word count of the processed data will be displayed. Output is stored in `results/esk610_spark_streaming/data/v0/dstream/wordcount`. Only new files in /tmp/eskapade_stream_test are processed, do not forget to delete this directory.

TCP stream

The word count example using the TCP stream method can be run by executing in two different terminals:

```
terminal 1 $ eskapade_run -c stream_type='tcp' python/eskapadespark/tutorials/esk610_
↪spark_streaming_wordcount.py

terminal 2 $ nc -lk 9999
```

Where `nc` (netcat) will stream data to port 9999 and Spark Streaming will listen to this port and process incoming data. In `terminal 2` random words can be type (followed by enter) and in `terminal 1` a word count of the processed data will by displayed. Output is stored in `results/esk610_spark_streaming/data/v0/dstream/wordcount`.

5.2 Release notes

5.2.1 Version 0.9

Eskapade-Spark v0.9 (December 2018) contains only one update compared with v0.8:

- All code has been updated to Eskapade v0.9, where the core functionality has been split off into the Eskapade-Core package. As such the code is backwards-incompatible with v0.8.

5.2.2 Version 0.8

Version 0.8 of Eskapade-Spark (August 2018) is a split off of the `spark-analysis` module of Eskapade v0.7 into a separate package. This way, Eskapade v0.8 no longer depends on Spark. This new package Eskapade-Spark does require Spark to install, clearly.

In addition, we have included new analysis code for processing ("flattening") time-series data, so it can be easily used as input for machine learning models. See tutorial example esk611 for details.

5.3 Developing and Contributing

5.3.1 Working on Eskapade-Spark

You have some cool feature and/or algorithm you want to add to Eskapade-Spark. How do you go about it?

First clone Eskapade-Spark.

```
git clone https://github.com/KaveIO/Eskapade-Spark.git eskapade-spark
```

then

```
pip install -e eskapade-spark
```

this will install Eskapade in editable mode, which will allow you to edit the code and run it as you would with a normal installation of eskapade.

To make sure that everything works try executing eskapade without any arguments, e.g.

```
eskapade_run --help
```

or you could just execute the tests using either the eskapade test runner, e.g.

```
eskapade_trial .
```

That's it.

5.3.2 Contributing

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change. You can find the contact information on the index page.

Note that when contributing that all tests should succeed.

5.4 References

- Web page: https://eskapade-spark.readthedocs.io
- Repository: https://github.com/kaveio/eskapade-root
- Issues & Ideas: https://github.com/kaveio/eskapade-spark/issues
- Eskapade: http://eskapade.kave.io
- Contact us at: kave [at] kpmg [dot] com

5.5 API

5.5.1 API Documentation

EskapadeSpark

eskapadespark package

Subpackages

eskapadespark.links package

Submodules

eskapadespark.links.daily_summary module

Project: Eskapade - A python-based package for data analysis.

Class: ExampleLink

Created: 2018-03-08

Description: Each feature given from the input df will by default correspond to 6 columns in the output: min, mean, max, stddev, count, and sum. The columns are named like 'feature_stddev_0d' (0d since we look 0 days back into the past).

The new dataframe will also contain the column *new_date_col* with the date, and all the identifying columns given in *partitionby_cols*.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.daily_summary.**DailySummary**(***kwargs*)

Bases: escore.core.element.Link

Creates daily summary information from a timeseries dataframe.

Each feature given from the input df will by default correspond to 6 columns in the output: min, mean, max, stddev, count, and sum. The columns are named like 'feature_stddev_0d' (0d since we look 0 days back into the past).

The new dataframe will also contain the column *new_date_col* with the date, and all the identifying columns given in *partitionby_cols*.

__init__(***kwargs*)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **feature_cols** (*list/dict*) – columns to take daily aggregates of. If list, all columns in the list are aggregated with the min, mean, max, stddev, count, and sum. If dict, the keys are column names to aggregate, and the values are lists of aggregation functions to apply. These must be built in spark aggregation functions.
- **new_date_col** (*str*) – name of the 'date' column which will be created (default 'date')
- **datetime_col** (*str*) – name of column with datetime information in the dataframe
- **partitionby_cols** (*list*) – identifying columns to partition by before aggregating

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

eskapadespark.links.find_days_until_event module

Project: Eskapade - A python-based package for data analysis.

Class: ExampleLink

Created: 2018-03-08

Description: Will create a new column (name given by *countdown_col_name*) containing the number of days between the current row and the next date on which *event_col* is greater than 0. The dataframe must include a column that has a date or datetime.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.find_days_until_event.**FindDaysUntilEvent**(***kwargs*)
Bases: escore.core.element.Link

Find the number of days until an event in a spark dataframe.

Will create a new column (name given by *countdown_col_name*) containing the number of days between the current row and the next date on which *event_col* is greater than 0. The dataframe must include a column that has a date or datetime.

__init__(***kwargs*)
Find the number of days until a particular event in an ordered dataframe.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **datetime_col** (*str*) – column with datetime information
- **event_col** (*str*) – the column containing the events (0 for rows with no events, >0 otherwise)
- **countdown_col_name** (*str*) – column where the number of days until the next event will be stored
- **partitionby_cols** (*str*) – columns to partition the countdown by

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

eskapadespark.links.rdd_group_mapper module

Project: Eskapade - A python-based package for data analysis.

Class: RddGroupMapper

Created: 2017/06/20

Description: Apply a map function on groups in a Spark RDD

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.rdd_group_mapper.**RddGroupMapper**(***kwargs*)
Bases: escore.core.element.Link

Apply a map function on groups in a Spark RDD.

Group rows of key-value pairs in a Spark RDD by key and apply a custom map function on the group values. By default, the group key and the value returned by the map function forms a single row in the output RDD. If the "flatten_output_groups" flag is set, the returned value is interpreted as an iterable and a row is created for each item.

Optionally, a map function is applied on the rows of the input RDD, for example to create the group key-value pairs. Similarly, a function may be specified to map the key-value pairs resulting from the group map.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **group_map** – map function for group values
- **input_map** – map function for input rows; optional, e.g. to create group key-value pairs
- **result_map** – map function for output group values; optional, e.g. to flatten group key-value pairs
- **flatten_output_groups** (*bool*) – create a row for each item in the group output values (default is False)
- **num_group_partitions** (*int*) – number of partitions for group map (optional, no repartitioning by default)

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_configurator module

Project: Eskapade - A python-based package for data analysis.

Class: SparkConfigurator

Created: 2017/06/07

Description: This link stops a running Spark session and starts a new one with the configuration provided to the link.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_configurator.**SparkConfigurator**(***kwargs*)
Bases: escore.core.element.Link

Set configuration settings of SparkContext.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **spark_settings** (*iterable*) – list of key/value pairs specifying the Spark configuration
- **log_level** (*str*) – verbosity level of the SparkContext

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_data_to_csv module

Project: Eskapade - A python-based package for data analysis.

Class : SparkDataToCsv

Created: 2015-11-16

Description: Write Spark data to local CSV files

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class eskapadespark.links.spark_data_to_csv.**SparkDataToCsv**(***kwargs*)
Bases: escore.core.element.Link

Write Spark data to local CSV files.

Data to write to CSV are provided as a Spark RDD or a Spark data frame. The data are written to a configurable number of CSV files in the specified output directory.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link instance
- **read_key** (*str*) – data-store key of the Spark data
- **output_path** (*str*) – directory path of the output CSV file(s)
- **mode** (*str*) – write mode if data already exist (“overwrite”, “ignore”, “error”)
- **compression_codec** (*str*) – compression-codec class (e.g., ‘org.apache.hadoop.io.compress.GzipCodec’)
- **sep** (*str*) – CSV separator string
- **header** (*tuple|bool*) – column names to write as CSV header or boolean to indicate if names must be determined from input data frame
- **num_files** (*int*) – requested number of output files

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_df_converter module

Project: Eskapade - A python-based package for data analysis.

Class: SparkDfConverter

Created: 2017/06/15

Description: Convert a Spark data frame into data of a different format

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_df_converter.**SparkDfConverter**(***kwargs*)
Bases: escore.core.element.Link

Link to convert a Spark data frame into a different format.

A data frame from the data store is converted into data of a different format and/or transformed. The format conversion is controlled by the "output_format" argument. The data frame can either be unchanged ("df", default) or converted into a Spark RDD of tuples ("RDD"), a list of tuples ("list"), or a Pandas data frame ("pd").

After the format conversion, the data can be transformed by functions specified by the "process_methods" argument. These functions will be sequentially applied to the output of the previous function. Each function is specified by either a callable object or a string. A string will be interpreted as the name of an attribute of the dataset type.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **schema_key** (*str*) – key to store the data-frame schema in the data store
- **output_format** (*str*) – data format to store: {"df" (default), "RDD", "list", "pd"}
- **preserve_col_names** (*bool*) – preserve column names for non-data-frame output formats (default is True)
- **process_methods** (*iterable*) – methods to apply sequentially on the produced data
- **process_meth_args** (*dict*) – positional arguments for process methods
- **process_meth_kwargs** (*dict*) – keyword arguments for process methods
- **fail_missing_data** (*bool*) – fail execution if the input data frame is missing (default is "True")

execute()
Execute the link.

initialize()
Initialize SparkDfConverter.

eskapadespark.links.spark_df_creator module

Project: Eskapade - A python-based package for data analysis.

Class: SparkDfCreator

Created: 2017/06/13

Description: Create a Spark data frame from generic input data

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class eskapadespark.links.spark_df_creator.**SparkDfCreator**(***kwargs*)
Bases: escore.core.element.Link

Link to create a Spark dataframe from generic input data.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **schema** – schema to create data frame if input data have a different format
- **process_methods** (*iterable*) – methods to apply sequentially on the produced data frame
- **process_meth_args** (*dict*) – positional arguments for process methods
- **process_meth_kwargs** (*dict*) – keyword arguments for process methods
- **fail_missing_data** (*bool*) – fail execution if data are missing (default is "True")

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_df_reader module

Project: Eskapade - A python-based package for data analysis.

Class: SparkDfReader

Created: 2016/11/08

Description: Read data into a Spark data frame

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_df_reader.**SparkDfReader**(***kwargs*)
Bases: escore.core.element.Link

Link to read data into a Spark dataframe.

Data are read with the Spark-SQL data-frame reader (pyspark.sql.DataFrameReader). The read-method to be applied on the reader instance (load, parquet, csv, ...) can be specified by the user, including its arguments. In addition to the read method, also other functions to be applied on the reader (schema, option, ...) and/or the resulting data frame (filter, select, repartition, ...) can be included.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **read_methods** (*iterable*) – methods to apply sequentially on data-frame reader and data frame
- **read_meth_args** (*dict*) – positional arguments for read methods
- **read_meth_kwargs** (*dict*) – keyword arguments for read methods

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_df_writer module

Project: Eskapade - A python-based package for data analysis.

Class: SparkDfWriter

Created: 2016/11/08

Description: Write data from a Spark data frame

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_df_writer.**SparkDfWriter**(***kwargs*)
Bases: escore.core.element.Link

Link to write data from a Spark dataframe.

Data are written with the Spark-SQL data-frame writer (pyspark.sql.DataFrameWriter). The write method to be applied (save, parquet, csv, ...) can be specified by the user, including its arguments. In addition to the write method, also other functions to be applied on the writer (format, option, ...) can be included.

If the input format is not a Spark data frame, an attempt is made to convert to a data frame. This works for lists, Spark RDDs, and Pandas data frames. A schema may be specified for the created data frame.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data in data store
- **schema** – schema to create data frame if input data have a different format
- **write_methods** (*iterable*) – methods to apply sequentially on data-frame writer
- **write_meth_args** (*dict*) – positional arguments for write methods
- **write_meth_kwargs** (*dict*) – keyword arguments for write methods
- **num_files** (*int*) – requested number of output files
- **fail_missing_data** (*bool*) – fail execution if data are missing (default is "True")

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_execute_query module

Project: Eskapade - A python-based package for data analysis.

Class: SparkExecuteQuery

Created: 2017/11/08

Description: SparkExecuteQuery applies a SQL-query to one or more objects in the DataStore and adds the output of the query to the DataStore as a Spark dataframe, RDD or Pandas dataframe.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_execute_query.**SparkExecuteQuery**(***kwargs*)
Bases: escore.core.element.Link

Defines the content of link SparkExecuteQuery.

Applies a SQL-query to one or more objects in the DataStore. Such SQL-queries can for instance be used to filter Spark dataframes. All objects in the DataStore are registered as SQL temporary views. The output of the query can be added to the DataStore as a Spark dataframe (default), RDD or Pandas dataframe.

__init__(***kwargs*)
Store the configuration of link SparkExecuteQuery.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **output_format** (*str*) – data format to store: {"df" (default), "rdd", "pd"}
- **query** (*str*) – a string containing a SQL-query.

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.spark_histogrammar_filler module

Project: Eskapade - A python-based package for data analysis.

Class: SparkHistogrammarFiller

Created: 2017/06/09

Description: Algorithm to fill histogrammar sparse-bin histograms from a Spark dataframe. It is possible to do cleaning of these histograms by rejecting certain keys or removing inconsistent data types. Timestamp columns are converted to nanoseconds before the binning is applied.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_histogrammar_filler.**SparkHistogrammarFiller**(***kwargs*)
Bases: eskapade.analysis.links.hist_filler.HistogrammarFiller

Fill histogrammar sparse-bin histograms with Spark.

Algorithm to fill histogrammar style sparse-bin and category histograms with Spark. It is possible to do after-filling cleaning of these histograms by rejecting certain keys or removing inconsistent data types. Timestamp columns are converted to nanoseconds before the binning is applied. Final histograms are stored in the datastore.

Example is available in: tutorials/esk605_hgr_filler_plotter.py.

__init__(***kwargs*)
Initialize link instance.

Store and do basic check on the attributes of link HistogrammarFiller.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store histograms in data store
- **columns** (*list*) – colums to pick up from input data (default is all columns)
- **bin_specs** (*dict*) – dictionaries used for rebinning numeric or timestamp columns

Example bin_specs dictionary is:

```
>>> bin_specs = {'x': {'bin_width': 1, 'bin_offset': 0},
>>>              'y': {'bin_edges': [0, 2, 3, 4, 5, 7, 8]}}
```

Parameters

- **var_dtype** (*dict*) – dict of datatypes of the columns to study from dataframe (if not provided, try to determine datatypes directy from dataframe)
- **quantity** (*dict*) – dictionary of lambda functions of how to pars certain columns

Example quantity dictionary is:

```
>>> quantity = {'y': lambda x: x}
```

Parameters

- **store_at_finalize** (*bool*) – store histograms in datastore at finalize(), not at execute() (useful when looping over datasets, default is False)
- **dict** (*drop_keys*) – dictionary used for dropping specific keys from bins dictionaries of histograms

Example drop_keys dictionary is:

```
>>> drop_keys = {'x': [1, 4, 8, 19],
>>               'y': ['apple', 'pear', 'tomato'],
>>               'x:y': [(1, 'apple'), (19, 'tomato')]}
```

assert_dataframe(*df*)
Check that input data is a filled Spark data frame.

Parameters **df** – input Spark data frame

construct_empty_hist(*df*, *columns*)
Create an (empty) histogram of right type.

Create a multi-dim histogram by iterating through the columns in reverse order and passing a single-dim hist as input to the next column.

Parameters

- **df** – input dataframe
- **columns** (*list*) – histogram columns

Returns created histogram

Return type histogrammar.Count

fill_histogram(*idf*, *columns*)
Fill input histogram with column(s) of input dataframe.

Parameters

- **idf** – input data frame used for filling histogram
- **columns** (*list*) – histogram column(s)

get_all_columns(*data*)
Retrieve all columns / keys from input data.

Parameters **data** – input data sample (pandas dataframe or dict)

Returns list of columns

Return type list

get_data_type(*df*, *col*)
Get data type of dataframe column.

Parameters

- **df** – input data frame
- **col** (*str*) – column

process_and_store()
Process and store spark-based histogram objects.

process_columns(*df*)
Process columns before histogram filling.

Specifically, in this case convert timestamp columns to nanoseconds

Parameters **df** – input data frame

Returns output data frame with converted timestamp columns

Return type DataFrame

eskapadespark.links.spark_histogrammar_filler.**hgr_patch_histogram**(*hist*)
Apply set of patches to histogrammer histogram.

Parameters **hist** – histogrammar histogram to patch up.

eskapadespark.links.spark_histogrammar_filler.**hgr_reset_quantity**(*hist*, *new_quantity=<function unit_func>*)
Reset quantity attribute of histogrammar histogram.

If quantity refers to a Spark df the histogram cannot be pickled, b/c we cannot pickle a Spark df. Here we reset the quantity of a (filled) histogram to a neutral lambda function.

Parameters

- **hist** – histogrammar histogram to reset quantity of.
- **new_quantity** – new quantity function to reset hist.quantity to. default is lambda x: x.

eskapadespark.links.spark_histogrammar_filler.**unit_func**(*x*)
Dummy quantity function for histogrammar objects

Parameters **x** – value

Returns the same value

eskapadespark.links.spark_streaming_controller module

Project: Eskapade - A python-based package for data analysis.

Class: SparkStreamingController

Created: 2017/07/12

Description: Link to start/stop Spark Stream.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_streaming_controller.**SparkStreamingController**(***kwargs*)
Bases: escore.core.element.Link

Defines the content of link SparkStreamingController.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link

- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **timeout** (*int*) – the amount of time (in seconds) for running the Spark Streaming Context

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

eskapadespark.links.spark_streaming_wordcount module

Project: Eskapade - A python-based package for data analysis.

Class: SparkStreamingWordCount

Created: 2017/07/12

Description: The Spark Streaming word count example derived from: https://spark.apache.org/docs/latest/streaming-programming-guide.html

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_streaming_wordcount.**SparkStreamingWordCount**(***kwargs*)
Bases: escore.core.element.Link

Counts words in UTF8 encoded delimited text.

Text is received from the network every second. To run this on your local machine, you need to first run a Netcat server

$ nc -lk 9999

and then run the example (in a second terminal)

$ eskapade_run tutorials/esk610_spark_streaming_wordcount.py

NB: hostname and port can be adapted in the macro.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

eskapadespark.links.spark_streaming_writer module

Project: Eskapade - A python-based package for data analysis.

Class: SparkStreamingWriter

Created: 2017/07/12

Description: This link writes Spark Stream DStream data to disk. The path specifies the directory on eithter local disk or HDFS where files are stored. Each processed RDD batch will be stored in a separate file (hence the number of files can increase rapidly).

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.links.spark_streaming_writer.**SparkStreamingWriter**(**kwargs*)
Bases: escore.core.element.Link

Link to write Spark Stream to disk.

__init__(**kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **path** (*str*) – the directory path of the output files (local disk or HDFS)
- **suffix** (*str*) – the suffix of the file names in the output directory
- **repartition** (*int*) – repartition RDD to number of files (default: single file per batch)

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

eskapadespark.links.spark_with_column module

Project: Eskapade - A python-based package for data analysis.

Class: SparkWithColumn

Created: 2018-03-08

Description: SparkWithColumn adds the output of a column expression (column operation, sql.functions function, or udf) to a dataframe.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class eskapadespark.links.spark_with_column.**SparkWithColumn**(***kwargs*)
Bases: escore.core.element.Link

Create a new column from columns in a Spark dataframe

SparkWithColumn adds the output of a column expression (column operation, sql.functions function, or udf) to a dataframe.

__init__(***kwargs*)
Initialize SparkWithColumn instance

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **store_key** (*str*) – key of data to store in data store
- **new_col_name** (*str*) – name of newly created column
- **new_col** (*Column*) – the column object to be included in the dataframe, resulting from a column expression

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

eskapadespark.links.sparkgeneralfuncprocessor module

Project: Eskapade - A python-based package for data analysis.

Class: SparkGeneralFuncProcessor

Created: 2016/11/08

Description: Processor for applying pandas function on a Spark dataframe.

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class eskapadespark.links.sparkgeneralfuncprocessor.**SparkGeneralFuncProcessor**(***kwargs*)
Bases: escore.core.element.Link

Processor for applying pandas function on a Spark dataframe.

The spark API is not (yet) as rich as the pandas API. Therefore sometimes one needs pandas to implement the desired algorithm. This link defines a general approach for applying an advanced function using pandas on a Spark dataframe. The Spark dataframe is grouped and the general function is applied on each group in parallel. In the general function a pandas dataframe can be created as follows: pandas_df = pd.DataFrame(list(group), columns=cols) For examples, see the function in the deutils.analysishelper module

This Link uses pyspark.RDD.groupByKey() function instead of pyspark.RDD.reduceBeKey() because one needs all the data of one group on one datanode in order to make a pandas dataframe from the group.

__init__(***kwargs*)
Initialize link instance.

Store the configuration of link SparkToGeneralFuncProcessor.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store. It should contain a spark dataframe or spark rdd.
- **store_key** (*str*) – key of data to store in data store
- **groupby** (*list*) – spark dataframe columns to group by
- **columns** (*list*) – The columns of the spark dataframe or RDD. Obligatory for RDD, not for spark dataframe.
- **generalfunc** (*func*) – The general function. Should be defined by the user. Arguments should be list of tuples (rows of RDD), column names and if necessary keyword arguments. Should return a list of native python types.
- **function_args** (*dict*) – Keyword arguments for the function
- **nb_partitions** (*int*) – The number of partitions for repartitioning after groupByKey
- **return_map** (*func*) – Function used by the map on the RDD after the generalfunc is applied. The default return a tuple of the groupby columns (row[0]) and the list returned by the generalfunc (row[1]).

execute()
Execute the link.

initialize()
Initialize the link.

eskapadespark.links.sparkhister module

Project: Eskapade - A python-based package for data analysis.

Class: SparkHister

Created: 2016/11/08

Description: Algorithm to do... (fill in here)

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

Redistribution and use in source and binary forms, with or without modification, are permitted according to the terms listed in the file LICENSE.

class eskapadespark.links.sparkhister.**SparkHister**(*name='HiveHister'*)
Bases: escore.core.element.Link

Defines the content of link SparkHister.

__init__(*name='HiveHister'*)
Initialize link instance.

Store the configuration of link SparkHister.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **store_key** (*str*) – key of data to store in data store
- **columns** (*list*) – columns of the Spark dataframe to make a histogram from
- **bins** (*dict*) – the bin edges of the histogram
- **convert_for_mongo** (*bool*) – if True the data structure of the result is converted so it can be stored in mongo

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

Module contents

class eskapadespark.links.**RddGroupMapper**(***kwargs*)
Bases: escore.core.element.Link

Apply a map function on groups in a Spark RDD.

Group rows of key-value pairs in a Spark RDD by key and apply a custom map function on the group values. By default, the group key and the value returned by the map function forms a single row in the output RDD. If the "flatten_output_groups" flag is set, the returned value is interpreted as an iterable and a row is created for each item.

Optionally, a map function is applied on the rows of the input RDD, for example to create the group key-value pairs. Similarly, a function may be specified to map the key-value pairs resulting from the group map.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **group_map** – map function for group values

- **input_map** – map function for input rows; optional, e.g. to create group key-value pairs
- **result_map** – map function for output group values; optional, e.g. to flatten group key-value pairs
- **flatten_output_groups** (*bool*) – create a row for each item in the group output values (default is False)
- **num_group_partitions** (*int*) – number of partitions for group map (optional, no repartitioning by default)

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkConfigurator**(***kwargs*)
Bases: escore.core.element.Link

Set configuration settings of SparkContext.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **spark_settings** (*iterable*) – list of key/value pairs specifying the Spark configuration
- **log_level** (*str*) – verbosity level of the SparkContext

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkDataToCsv**(***kwargs*)
Bases: escore.core.element.Link

Write Spark data to local CSV files.

Data to write to CSV are provided as a Spark RDD or a Spark data frame. The data are written to a configurable number of CSV files in the specified output directory.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link instance
- **read_key** (*str*) – data-store key of the Spark data
- **output_path** (*str*) – directory path of the output CSV file(s)
- **mode** (*str*) – write mode if data already exist ("overwrite", "ignore", "error")
- **compression_codec** (*str*) – compression-codec class (e.g., 'org.apache.hadoop.io.compress.GzipCodec')
- **sep** (*str*) – CSV separator string

- **header** (*tuple|bool*) – column names to write as CSV header or boolean to indicate if names must be determined from input data frame
- **num_files** (*int*) – requested number of output files

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkDfConverter**(***kwargs*)
Bases: escore.core.element.Link

Link to convert a Spark data frame into a different format.

A data frame from the data store is converted into data of a different format and/or transformed. The format conversion is controlled by the "output_format" argument. The data frame can either be unchanged ("df", default) or converted into a Spark RDD of tuples ("RDD"), a list of tuples ("list"), or a Pandas data frame ("pd").

After the format conversion, the data can be transformed by functions specified by the "process_methods" argument. These functions will be sequentially applied to the output of the previous function. Each function is specified by either a callable object or a string. A string will be interpreted as the name of an attribute of the dataset type.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **schema_key** (*str*) – key to store the data-frame schema in the data store
- **output_format** (*str*) – data format to store: {"df" (default), "RDD", "list", "pd"}
- **preserve_col_names** (*bool*) – preserve column names for non-data-frame output formats (default is True)
- **process_methods** (*iterable*) – methods to apply sequentially on the produced data
- **process_meth_args** (*dict*) – positional arguments for process methods
- **process_meth_kwargs** (*dict*) – keyword arguments for process methods
- **fail_missing_data** (*bool*) – fail execution if the input data frame is missing (default is "True")

execute()
Execute the link.

initialize()
Initialize SparkDfConverter.

class eskapadespark.links.**SparkDfCreator**(***kwargs*)
Bases: escore.core.element.Link

Link to create a Spark dataframe from generic input data.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of the input data in the data store
- **store_key** (*str*) – key of the output data frame in the data store
- **schema** – schema to create data frame if input data have a different format
- **process_methods** (*iterable*) – methods to apply sequentially on the produced data frame
- **process_meth_args** (*dict*) – positional arguments for process methods
- **process_meth_kwargs** (*dict*) – keyword arguments for process methods
- **fail_missing_data** (*bool*) – fail execution if data are missing (default is "True")

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkDfReader**(**kwargs*)
Bases: escore.core.element.Link

Link to read data into a Spark dataframe.

Data are read with the Spark-SQL data-frame reader (pyspark.sql.DataFrameReader). The read-method to be applied on the reader instance (load, parquet, csv, ...) can be specified by the user, including its arguments. In addition to the read method, also other functions to be applied on the reader (schema, option, ...) and/or the resulting data frame (filter, select, repartition, ...) can be included.

__init__(**kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **read_methods** (*iterable*) – methods to apply sequentially on data-frame reader and data frame
- **read_meth_args** (*dict*) – positional arguments for read methods
- **read_meth_kwargs** (*dict*) – keyword arguments for read methods

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkDfWriter**(**kwargs*)
Bases: escore.core.element.Link

Link to write data from a Spark dataframe.

Data are written with the Spark-SQL data-frame writer (pyspark.sql.DataFrameWriter). The write method to be applied (save, parquet, csv, ...) can be specified by the user, including its arguments. In addition to the write method, also other functions to be applied on the writer (format, option, ...) can be included.

If the input format is not a Spark data frame, an attempt is made to convert to a data frame. This works for lists, Spark RDDs, and Pandas data frames. A schema may be specified for the created data frame.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data in data store
- **schema** – schema to create data frame if input data have a different format
- **write_methods** (*iterable*) – methods to apply sequentially on data-frame writer
- **write_meth_args** (*dict*) – positional arguments for write methods
- **write_meth_kwargs** (*dict*) – keyword arguments for write methods
- **num_files** (*int*) – requested number of output files
- **fail_missing_data** (*bool*) – fail execution if data are missing (default is "True")

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkExecuteQuery**(***kwargs*)
Bases: escore.core.element.Link

Defines the content of link SparkExecuteQuery.

Applies a SQL-query to one or more objects in the DataStore. Such SQL-queries can for instance be used to filter Spark dataframes. All objects in the DataStore are registered as SQL temporary views. The output of the query can be added to the DataStore as a Spark dataframe (default), RDD or Pandas dataframe.

__init__(***kwargs*)
Store the configuration of link SparkExecuteQuery.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **output_format** (*str*) – data format to store: {"df" (default), "rdd", "pd"}
- **query** (*str*) – a string containing a SQL-query.

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkHistogrammarFiller**(***kwargs*)
Bases: eskapade.analysis.links.hist_filler.HistogrammarFiller

Fill histogrammar sparse-bin histograms with Spark.

Algorithm to fill histogrammar style sparse-bin and category histograms with Spark. It is possible to do after-filling cleaning of these histograms by rejecting certain keys or removing inconsistent data types. Timestamp columns are converted to nanoseconds before the binning is applied. Final histograms are stored in the datastore.

Example is available in: tutorials/esk605_hgr_filler_plotter.py.

__init__(***kwargs*)
Initialize link instance.

Store and do basic check on the attributes of link HistogrammarFiller.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store histograms in data store
- **columns** (*list*) – colums to pick up from input data (default is all columns)
- **bin_specs** (*dict*) – dictionaries used for rebinning numeric or timestamp columns

Example bin_specs dictionary is:

```
>>> bin_specs = {'x': {'bin_width': 1, 'bin_offset': 0},
>>>              'y': {'bin_edges': [0, 2, 3, 4, 5, 7, 8]}}
```

Parameters

- **var_dtype** (*dict*) – dict of datatypes of the columns to study from dataframe (if not provided, try to determine datatypes directy from dataframe)
- **quantity** (*dict*) – dictionary of lambda functions of how to pars certain columns

Example quantity dictionary is:

```
>>> quantity = {'y': lambda x: x}
```

Parameters

- **store_at_finalize** (*bool*) – store histograms in datastore at finalize(), not at execute() (useful when looping over datasets, default is False)
- **dict** (*drop_keys*) – dictionary used for dropping specific keys from bins dictionaries of histograms

Example drop_keys dictionary is:

```
>>> drop_keys = {'x': [1, 4, 8, 19],
>>>              'y': ['apple', 'pear', 'tomato'],
>>>              'x:y': [(1, 'apple'), (19, 'tomato')]}
```

assert_dataframe(*df*)
Check that input data is a filled Spark data frame.

Parameters **df** – input Spark data frame

construct_empty_hist(*df*, *columns*)
Create an (empty) histogram of right type.

Create a multi-dim histogram by iterating through the columns in reverse order and passing a single-dim hist as input to the next column.

Parameters

- **df** – input dataframe

- **columns** (*list*) – histogram columns

Returns created histogram

Return type histogrammar.Count

fill_histogram(*idf*, *columns*)
Fill input histogram with column(s) of input dataframe.

Parameters

- **idf** – input data frame used for filling histogram
- **columns** (*list*) – histogram column(s)

get_all_columns(*data*)
Retrieve all columns / keys from input data.

Parameters **data** – input data sample (pandas dataframe or dict)

Returns list of columns

Return type list

get_data_type(*df*, *col*)
Get data type of dataframe column.

Parameters

- **df** – input data frame
- **col** (*str*) – column

process_and_store()
Process and store spark-based histogram objects.

process_columns(*df*)
Process columns before histogram filling.

Specifically, in this case convert timestamp columns to nanoseconds

Parameters **df** – input data frame

Returns output data frame with converted timestamp columns

Return type DataFrame

class eskapadespark.links.**SparkStreamingController**(***kwargs*)
Bases: escore.core.element.Link

Defines the content of link SparkStreamingController.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **timeout** (*int*) – the amount of time (in seconds) for running the Spark Streaming Context

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkStreamingWordCount**(**kwargs)
Bases: escore.core.element.Link

Counts words in UTF8 encoded delimited text.

Text is received from the network every second. To run this on your local machine, you need to first run a Netcat server

$ nc -lk 9999

and then run the example (in a second terminal)

$ eskapade_run tutorials/esk610_spark_streaming_wordcount.py

NB: hostname and port can be adapted in the macro.

__init__(**kwargs)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkStreamingWriter**(**kwargs)
Bases: escore.core.element.Link

Link to write Spark Stream to disk.

__init__(**kwargs)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **path** (*str*) – the directory path of the output files (local disk or HDFS)
- **suffix** (*str*) – the suffix of the file names in the output directory
- **repartition** (*int*) – repartition RDD to number of files (default: single file per batch)

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkWithColumn**(***kwargs*)
Bases: escore.core.element.Link

Create a new column from columns in a Spark dataframe

SparkWithColumn adds the output of a column expression (column operation, sql.functions function, or udf) to a dataframe.

__init__(***kwargs*)
Initialize SparkWithColumn instance

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **store_key** (*str*) – key of data to store in data store
- **new_col_name** (*str*) – name of newly created column
- **new_col** (*Column*) – the column object to be included in the dataframe, resulting from a column expression

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

class eskapadespark.links.**SparkGeneralFuncProcessor**(***kwargs*)
Bases: escore.core.element.Link

Processor for applying pandas function on a Spark dataframe.

The spark API is not (yet) as rich as the pandas API. Therefore sometimes one needs pandas to implement the desired algorithm. This link defines a general approach for applying an advanced function using pandas on a Spark dataframe. The Spark dataframe is grouped and the general function is applied on each group in parallel. In the general function a pandas dataframe can be created as follows: pandas_df = pd.DataFrame(list(group), columns=cols) For examples, see the function in the deutils.analysishelper module

This Link uses pyspark.RDD.groupByKey() function instead of pyspark.RDD.reduceBeKey() because one needs all the data of one group on one datanode in order to make a pandas dataframe from the group.

__init__(***kwargs*)
Initialize link instance.

Store the configuration of link SparkToGeneralFuncProcessor.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store. It should contain a spark dataframe or spark rdd.
- **store_key** (*str*) – key of data to store in data store
- **groupby** (*list*) – spark dataframe columns to group by
- **columns** (*list*) – The columns of the spark dataframe or RDD. Obligatory for RDD, not for spark dataframe.
- **generalfunc** (*func*) – The general function. Should be defined by the user. Arguments should be list of tuples (rows of RDD), column names and if necessary keyword arguments. Should return a list of native python types.
- **function_args** (*dict*) – Keyword arguments for the function
- **nb_partitions** (*int*) – The number of partitions for repartitioning after groupByKey
- **return_map** (*func*) – Function used by the map on the RDD after the generalfunc is applied. The default return a tuple of the groupby columns (row[0]) and the list returned by the generalfunc (row[1]).

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**SparkHister**(*name='HiveHister'*)
Bases: escore.core.element.Link

Defines the content of link SparkHister.

__init__(*name='HiveHister'*)
Initialize link instance.

Store the configuration of link SparkHister.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **store_key** (*str*) – key of data to store in data store
- **columns** (*list*) – columns of the Spark dataframe to make a histogram from
- **bins** (*dict*) – the bin edges of the histogram
- **convert_for_mongo** (*bool*) – if True the data structure of the result is converted so it can be stored in mongo

execute()
Execute the link.

finalize()
Finalize the link.

initialize()
Initialize the link.

class eskapadespark.links.**DailySummary**(**kwargs)
Bases: escore.core.element.Link

Creates daily summary information from a timeseries dataframe.

Each feature given from the input df will by default correspond to 6 columns in the output: min, mean, max, stddev, count, and sum. The columns are named like 'feature_stddev_0d' (0d since we look 0 days back into the past).

The new dataframe will also contain the column *new_date_col* with the date, and all the identifying columns given in *partitionby_cols*.

__init__(**kwargs)
Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **feature_cols** (*list/dict*) – columns to take daily aggregates of. If list, all columns in the list are aggregated with the min, mean, max, stddev, count, and sum. If dict, the keys are column names to aggregate, and the values are lists of aggregation functions to apply. These must be built in spark aggregation functions.
- **new_date_col** (*str*) – name of the 'date' column which will be created (default 'date')
- **datetime_col** (*str*) – name of column with datetime information in the dataframe
- **partitionby_cols** (*list*) – identifying columns to partition by before aggregating

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

class eskapadespark.links.**FindDaysUntilEvent**(**kwargs)
Bases: escore.core.element.Link

Find the number of days until an event in a spark dataframe.

Will create a new column (name given by *countdown_col_name*) containing the number of days between the current row and the next date on which *event_col* is greater than 0. The dataframe must include a column that has a date or datetime.

__init__(***kwargs*)
Find the number of days until a particular event in an ordered dataframe.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **store_key** (*str*) – key of output data to store in data store
- **datetime_col** (*str*) – column with datetime information
- **event_col** (*str*) – the column containing the events (0 for rows with no events, >0 otherwise)
- **countdown_col_name** (*str*) – column where the number of days until the next event will be stored
- **partitionby_cols** (*str*) – columns to partition the countdown by

execute()
Execute the link.

Returns status code of execution

Return type StatusCode

finalize()
Finalize the link.

Returns status code of finalization

Return type StatusCode

initialize()
Initialize the link.

Returns status code of initialization

Return type StatusCode

class eskapadespark.links.**HiveReader**(***kwargs*)
Bases: escore.core.element.Link

Link to read a Hive table into a Spark dataframe.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **databaseName** (*str*) – name of the hive database
- **tableName** (*str*) – name of the hive table
- **store_key** (*str*) – key of data to store in data store
- **columns** (*list*) – hive columns to read. If empty all columns will be queried.
- **selection** (*str*) – where clause of the hive query
- **limit** (*str*) – limit clause of the hive query
- **processFuncs** (*dict*) – process spark functions after query
- **full_query** (*str*) – if not empty execute only this querystring

- **hive_sql_file** (*str*) – path to an hive.sql file. If not empty the query in this file will be executed

add_proc_func(*func*, ***kwargs*)

execute()
Execute the link.

initialize()
Initialize the link.

class eskapadespark.links.**HiveWriter**(***kwargs*)
Bases: escore.core.element.Link

Link to write a dataframe in the datastore into a Hive table.

__init__(***kwargs*)
Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from data store
- **db** (*str*) – hive database name
- **table** (*str*) – hive table name
- **schemSpec** (*dict*) – if writing spark rdd, schema of hive types
- **prefix** (*str*) – prefix for hive column names
- **column_names_not_to_change** (*list*) – column names not to give the prefix
- **columns** (*list*) – columns to store in hive. If empty all columns will be stored
- **not_columns** (*list*) – columns to store not in hive
- **change_column_names** (*list*) – columns only to add prefix to

execute()
Execute the link.

initialize()
Initialize the link.

Submodules

eskapadespark.data_conversion module

Project: Eskapade - A python-based package for data analysis.

Module: spark_analysis.data_conversion

Created: 2017/05/30

Description: Converters between Spark, Pandas, and Python data formats

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

eskapadespark.data_conversion.**create_spark_df**(*spark*, *data*, *schema=None*, *process_methods=None*, ***kwargs*)

Create a Spark data frame from data in a different format.

A Spark data frame is created with either a specified schema or a schema inferred from the input data. The schema can be specified with the keyword argument "schema".

Functions to transform the data frame after creation can be specified by the keyword argument "process_methods". The value of this argument is an iterable of (function, arguments, keyword arguments) tuples to apply.

The data frame is created with the createDataFrame function of the SparkSession. Remaining keyword arguments are passed to this function.

```
>>> spark = pyspark.sql.SparkSession.builder.getOrCreate()
>>> df = create_spark_df(spark,
>>>                      [[1, 1.1, 'one'], [2, 2.2, 'two']],
>>>                      schema=['int', 'float', 'str'],
>>>                      process_methods=[('repartition', (), {'numPartitions': 6}
↪)])
>>> df.show()
+---+-----+---+
|int|float|str|
+---+-----+---+
|  2|  2.2|two|
|  1|  1.1|one|
+---+-----+---+
```

Parameters

- **spark** (*pyspark.sql.SparkSession*) – SparkSession instance
- **data** – input dataset
- **schema** – schema of created data frame
- **process_methods** (*iterable*) – methods to apply on the data frame after creation

Returns created data frame

Return type pyspark.sql.DataFrame

eskapadespark.data_conversion.**df_schema**(*schema_spec*)

Create Spark data-frame schema.

Create a schema for a Spark data frame from a dictionary of (name, data type) pairs, describing the columns. Data types are specified by Python types or by Spark-SQL types from the pyspark.sql.types module.

```
>>> from collections import OrderedDict as odict
>>> schema_dict = odict()
>>> schema_dict['foo'] = pyspark.sql.types.IntegerType()
>>> schema_dict['bar'] = odict([('descr', str), ('val', float)])
>>> print(schema_dict)
OrderedDict([('foo', IntegerType), ('bar', OrderedDict([('descr', <class 'str'>),␣
↪('val', <class 'float'>)]))])
>>> spark = pyspark.sql.SparkSession.builder.getOrCreate()
>>> df = spark.createDataFrame([(1, ('one', 1.1)), (2, ('two', 2.2))], schema=df_
↪schema(schema_dict))
>>> df.show()
+---+---------+
```

(continues on next page)

(continued from previous page)

```
|foo|      bar|
+---+---------+
|  1|[one,1.1]|
|  2|[two,2.2]|
+---+---------+
```

Parameters **schema_spec** (*dict*) – schema specification

Returns data-frame schema

Return type pyspark.sql.types.StructType

Raises TypeError if data type is specified incorrectly

eskapadespark.data_conversion.**hive_table_from_df**(*spark*, *df*, *db*, *table*)

Create a Hive table from a Spark data frame.

Parameters

- **spark** (*pyspark.sql.SparkSession*) – SparkSession instance
- **df** (*pyspark.sql.DataFrame*) – input data frame
- **db** (*str*) – database for table
- **table** (*str*) – name of table

eskapadespark.decorators module

Project: Eskapade - A python-based package for data analysis.

Module: spark_analysis.decorators

Created: 2017/05/24

Description: Decorators for Spark objects

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

eskapadespark.decorators.**spark_cls_reduce**(*self*)

Reduce function for Spark classes.

Spark objects connected to distributed data cannot be stored in Pickle files. This custom reduce function enables Pickling of a string representation of the Spark object.

eskapadespark.exceptions module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/03/31

Description: Eskapade exceptions

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

exception eskapadespark.exceptions.**MissingJayDeBeApiError**(*message=''*, *required_by=''*)

Bases: escore.exceptions.MissingPackageError

Exception raised if JayDeBeAPI is missing.

__init__(*message=''*, *required_by=''*)

Set missing-package arguments.

Parameters

- **message** (*str*) – message to show when raised
- **required_by** (*str*) – info on component that requires the package

exception eskapadespark.exceptions.**MissingPy4jError**(*message=''*, *required_by=''*)

Bases: escore.exceptions.MissingPackageError

Exception raised if Py4J is missing.

__init__(*message=''*, *required_by=''*)

Set missing-package arguments.

Parameters

- **message** (*str*) – message to show when raised
- **required_by** (*str*) – info on component that requires the package

exception eskapadespark.exceptions.**MissingSparkError**(*message=''*, *required_by=''*)

Bases: escore.exceptions.MissingPackageError

Exception raised if Spark is missing.

__init__(*message=''*, *required_by=''*)

Set missing-package arguments.

Parameters

- **message** (*str*) – message to show when raised
- **required_by** (*str*) – info on component that requires the package

eskapadespark.functions module

Project: Eskapade - A python-based package for data analysis.

Module: spark_analysis.functions

Created: 2017/05/24

Description: Collection of Spark functions defined for Eskapade

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

eskapadespark.functions.**calc_asym**(*var1*, *var2*)

Calculate asymmetry.

Calculate asymmetry between variables 1 and 2: >>> (var2 - var1) / (abs(var1) + abs(var2))

Returns asymmetry value

Return type float

eskapadespark.functions.**is_inf**(*x*)
Test if value is infinite.

eskapadespark.functions.**is_nan**(*x*)
Test if value is NaN/null/None.

eskapadespark.functions.**spark_query_func**(*spec*)
Get Eskapade Spark-query function.

Get a function that returns a string to be used as a function in a Spark SQL query:

```
>>> count_fun = spark_query_func('count')
>>> count_fun()
'count(*)'
>>> cov_fun = spark_query_func('cov')
>>> cov_fun('x', 'y')
'covar_pop(if(is_nan(x) or is_inf(x), NULL, x),if(is_nan(y) or is_inf(y), NULL,␣
↪y))'
>>> my_fun = spark_query_func('my_func::count(if({0:s} == 0, 1, NULL))')
>>> my_fun.__name__
'my_func'
>>> my_fun('my_var')
'count(if(my_var == 0, 1, NULL))'
```

Parameters **spec** (*str*) – function specification: “name” or “name::definition”

Returns query function

eskapadespark.functions.**spark_sql_func**(*name*, *default_func=None*)
Get Spark SQL function.

Get a function from pyspark.sql.functions by name. If function does not exist in the SQL-functions module, return a default function, if specified.

Parameters

- **name** (*str*) – name of function
- **default_func** – default function

Returns Spark SQL function

Raises RuntimeError if function does not exist

eskapadespark.functions.**to_date_time**(*dt*, *tz_in=None*, *tz_out=None*)
Convert value to date/time object.

Parameters

- **dt** – value representing a date/time (parsed by pandas.Timestamp)
- **tz_in** – time zone to localize data/time value to (parsed by pandas.Timestamp.tz_localize)
- **tz_out** – time zone to convert data/time value into (parsed by pandas.Timestamp.tz_convert)

Returns date/time object

Return type datetime.datetime

eskapadespark.functions.**to_timestamp**(*dt*, *tz_in=None*)
Convert value to Unix timestamp (ns).

Parameters

- **dt** – value representing a date/time (parsed by pandas.Timestamp)
- **tz_in** – time zone to localize data/time value to (parsed by pandas.Timestamp.tz_localize)

Returns Unix timestamp (ns)

Return type int

eskapadespark.resources module

Used by autodoc_mock_imports.

eskapadespark.spark_manager module

Project: Eskapade - A python-based package for data analysis.

Created: 2017/02/27

Class: SparkManager

Description: Process service for managing Spark operations

Authors: KPMG Advanced Analytics & Big Data team, Amstelveen, The Netherlands

class eskapadespark.spark_manager.**SparkManager**(*config_path=None*)

Bases: escore.core.process_services.ProcessService, escore.core.mixin.ConfigMixin

Process service for managing Spark operations.

__init__(*config_path=None*)

Initialize Spark manager instance.

create_session(*enable_hive_support=False*, *include_eskapade_modules=False*, ***conf_kwargs*)

Get or create Spark session.

Return the Spark-session instance. Create the session if it does not exist yet. If no SparkConfig is set yet, it is created. All keyword arguments are passed to the _create_spark_conf method in this case.

Parameters

- **enable_hive_support** (*bool*) – switch for enabling Spark Hive support
- **include_eskapade_modules** (*bool*) – switch to include Eskapade modules in Spark job submission. Default is False. Optional.

finish()

Stop Spark session.

get_session()

Get Spark session.

Return running Spark session and check if the Spark context is still alive.

spark_streaming_context

Spark Streaming Context.

eskapadespark.version module

THIS FILE IS AUTO-GENERATED BY ESKAPADE SETUP.PY.

Module contents

5.6 Appendices

5.6.1 Miscellaneous

Collection of miscelleneous Eskapade related items.

- See Apache Spark for details on using Spark with Eskapade.

Apache Spark

Eskapade supports the use of Apache Spark for parallel processing of large data volumes. Jobs can run on a single laptop using Spark libraries as well as on a Spark/Hadoop cluster in combination with YARN. This section describes how to setup and configure Spark for use with Eskapade. For examples on running Spark jobs with Eskapade, see the Spark tutorial.

Note: Eskapade supports both batch and streaming processing with Apache Spark.

Requirements

A working setup of the Apache Spark libraries is included in both the Eskapade docker and vagrant image (see section Installation). For installation of Spark libraries in a custom setup, please refer to the Spark documentation.

Spark installation

The environment variables `SPARK_HOME` and `PYTHONPATH` need be set and to point to the location of the Spark installation and the Python libraries of Spark and `py4j` (dependency). In the Eskapade docker, for example, it is set to:

```
$ echo $SPARK_HOME
/opt/spark/pro/
$ echo $PYTHONPATH
/opt/spark/pro/python:/opt/spark/pro/python/lib/py4j-0.10.4-src.zip:...
```

Configuration

The Spark configuration can be set in two ways:

1. an Eskapade macro (preferred)
2. an Eskapade link

This is demonstrated in the following tutorial macro:

```
$ eskapade_run python/eskapade/tutorials/esk601_spark_configuration.py
```

Both methods are described below. For a full explanation of Spark configuration settings, see Spark Configuration. In case configuration settings seem not to be picked up correctly, please check *Notes* at the end of this section.

Eskapade macro (preferred)

This method allows to specify settings per macro, i.e. per analysis, and is therefore the preferred way for bookkeeping analysis-specific settings.

The most easy way to start a Spark session is:

```
from eskapade import process_manager
from eskapade.spark_analysis import SparkManager

spark = sm.create_session(eskapade_settings=settings)
sc = spark.sparkContext
```

The default Spark configuration file `python/eskapade/config/spark/spark.cfg` will be picked up. It contains the following settings:

```
[spark]
spark.app.name=es_spark
spark.jars.packages=org.diana-hep:histogrammar-sparksql_2.11:1.0.4
spark.master=local[*]
spark.driver.host=localhost
```

The default Spark settings can be adapted here for all macros at once. In case, alternative settings are only relevant for a single analysis, those settings can also be specified in the macro using the argument variables in the `create_session` method of the SparkManager:

```
from eskapade import process_manager
from eskapade.spark_analysis import SparkManager

spark = sm.create_session(spark_settings=[('spark.app.name', 'es_spark_alt_config'), (
↪'spark.master', 'local[42]')])

sm = process_manager.service(SparkManager)
spark = sm.create_session(eskapade_settings=settings,
                          spark_settings=spark_settings,
                          config_path='/path/to/alternative/spark.cfg',
                          enable_hive_support=False,
                          include_eskapade_modules=False
                         )
```

Where all arguments are optional:

- `eskapade_settings` default configuration file as specified by the `sparkCfgFile` key in ConfigObject (i.e. `spark.cfg`)
- `config_path` alternative path to configuration file
- `spark_settings` list of key-value pairs to specify additional Spark settings
- `enable_hive_support`: switch to disable/enable Spark Hive support
- `include_eskapade_modules`: switch to include/exclude Eskapade modules in Spark job submission (e.g. for user-defined functions)

Eskapade link

This method allows to (re-)start Spark sessions from within a `SparkConfigurator` link. This means that by specifying multiple instances of this link in a macro, multiple Spark sessions with different settings can sequentially be run. This can be useful for larger analysis jobs that contain multiple Spark queries with very different CPU/memory needs - although the recently introduced *Dynamic allocation* feature is a more elegant way to achieve this behaviour.

Configurations for Spark jobs are set via the `SparkConf` class that holds a list of key/value pairs with settings, e.g.:

```
from eskapade import Chain
from eskapade.spark_analysis import SparkConfigurator

conf_link = SparkConfigurator(name='SparkConfigurator', spark_settings=[('spark.master
↪', 'local[3]')])
conf_link.log_level = 'INFO'
config = Chain('Config')
config.add(conf_link)
```

Note that the `SparkConfigurator` stops any existing Spark session before starting a new one. This means that the user should make sure all relevant data is stored at this point, since all cached Spark data will be cleared from memory.

Parameters

The most important parameters to play with for optimal performance:

- `num-executors`
- `executor-cores`
- `executor-memory`
- `driver-memory`

Dynamic allocation

Since version 2.1, Spark allows for dynamic resouce allocation. This requires the following settings:

- `spark.dynamicAllocation.enabled=true`
- `spark.shuffle.service.enabled=true`

Depending on the mode (standalone, YARN, Mesos), an additional shuffle service needs to be set up. See the documentation for details.

Logging

The logging level of Spark can be controlled in two ways:

1. through `$SPARK_HOME/conf/log4j.properties`

```
log4j.logger.org.apache.spark.api.python.PythonGatewayServer=INFO
```

2. through the `SparkContext` in Python:

```
spark = process_manager.service(SparkManager).get_session()
spark.sparkContext.setLogLevel('INFO')
```

PS: the loggers in Python can be controlled through:

```
import logging
print(logging.Logger.manager.loggerDict) # obtain list of all registered loggers
logging.getLogger('py4j').setLevel('INFO')
logging.getLogger('py4j.java_gateway').setLevel('INFO')
```

However, not all Spark-related loggers are available here (as they are JAVA-based).

Notes

There are a few pitfalls w.r.t. setting up Spark correctly:

1. If the environment variable `PYSPARK_SUBMIT_ARGS` is defined, its settings may override those specified in the macro/link. This can be prevented by unsetting the variable:

```
$ unset PYSPARK_SUBMIT_ARGS
```

or in the macro:

```
import os
del os.environ['PYSPARK_SUBMIT_ARGS']
```

The former will clear the variable from the shell session, whereas the latter will only clear it in the Python session.

2. In client mode not all driver options set via `SparkConf` are picked up at job submission because the JVM has already been started. Those settings should therefore be passed through the `SPARK_OPTS` environment variable, instead of using `SparkConf` in an Eskapade macro or link:

```
SPARK_OPTS=--driver-java-options=-Xms1024M --driver-java-options=-Xmx4096M --driver-
↪java-options=-Dlog4j.logLevel=info --driver-memory 2g
```

3. In case a Spark machine is not connected to a network, setting the `SPARK_LOCAL_HOSTNAME` environment variable or the `spark.driver.host` key in `SparkConf` to the value `localhost` may fix DNS resolution timeouts which prevent Spark from starting jobs.

5.7 Indices and tables

- genindex
- modindex

Python Module Index

e

eskapadespark, 52
eskapadespark.data_conversion, 46
eskapadespark.decorators, 48
eskapadespark.exceptions, 48
eskapadespark.functions, 49
eskapadespark.links, 34
eskapadespark.links.daily_summary, 18
eskapadespark.links.find_days_until_event, 19
eskapadespark.links.rdd_group_mapper, 20
eskapadespark.links.spark_configurator, 21
eskapadespark.links.spark_data_to_csv, 22
eskapadespark.links.spark_df_converter, 23
eskapadespark.links.spark_df_creator, 24
eskapadespark.links.spark_df_reader, 24
eskapadespark.links.spark_df_writer, 25
eskapadespark.links.spark_execute_query, 26
eskapadespark.links.spark_histogrammar_filler, 27
eskapadespark.links.spark_streaming_controller, 29
eskapadespark.links.spark_streaming_wordcount, 30
eskapadespark.links.spark_streaming_writer, 31
eskapadespark.links.spark_with_column, 31
eskapadespark.links.sparkgeneralfuncprocessor, 32
eskapadespark.links.sparkhister, 33
eskapadespark.resources, 51
eskapadespark.spark_manager, 51
eskapadespark.version, 52

Index

Symbols

__init__() (eskapadespark.exceptions.MissingJayDeBeApiError method), 49
__init__() (eskapadespark.exceptions.MissingPy4jError method), 49
__init__() (eskapadespark.exceptions.MissingSparkError method), 49
__init__() (eskapadespark.links.DailySummary method), 44
__init__() (eskapadespark.links.FindDaysUntilEvent method), 44
__init__() (eskapadespark.links.HiveReader method), 45
__init__() (eskapadespark.links.HiveWriter method), 46
__init__() (eskapadespark.links.RddGroupMapper method), 34
__init__() (eskapadespark.links.SparkConfigurator method), 35
__init__() (eskapadespark.links.SparkDataToCsv method), 35
__init__() (eskapadespark.links.SparkDfConverter method), 36
__init__() (eskapadespark.links.SparkDfCreator method), 36
__init__() (eskapadespark.links.SparkDfReader method), 37
__init__() (eskapadespark.links.SparkDfWriter method), 38
__init__() (eskapadespark.links.SparkExecuteQuery method), 38
__init__() (eskapadespark.links.SparkGeneralFuncProcessor method), 42
__init__() (eskapadespark.links.SparkHister method), 43
__init__() (eskapadespark.links.SparkHistogrammarFiller method), 39
__init__() (eskapadespark.links.SparkStreamingController method), 40
__init__() (eskapadespark.links.SparkStreamingWordCount method), 41
__init__() (eskapadespark.links.SparkStreamingWriter method), 41
__init__() (eskapadespark.links.SparkWithColumn method), 42
__init__() (eskapadespark.links.daily_summary.DailySummary method), 19
__init__() (eskapadespark.links.find_days_until_event.FindDaysUntilEvent method), 20
__init__() (eskapadespark.links.rdd_group_mapper.RddGroupMapper method), 21
__init__() (eskapadespark.links.spark_configurator.SparkConfigurator method), 21
__init__() (eskapadespark.links.spark_data_to_csv.SparkDataToCsv method), 22
__init__() (eskapadespark.links.spark_df_converter.SparkDfConverter method), 23
__init__() (eskapadespark.links.spark_df_creator.SparkDfCreator method), 24
__init__() (eskapadespark.links.spark_df_reader.SparkDfReader method), 25
__init__() (eskapadespark.links.spark_df_writer.SparkDfWriter method), 25
__init__() (eskapadespark.links.spark_execute_query.SparkExecuteQuery method), 26
__init__() (eskapadespark.links.spark_histogrammar_filler.SparkHistogram method), 27
__init__() (eskapadespark.links.spark_streaming_controller.SparkStreaming method), 29
__init__() (eskapadespark.links.spark_streaming_wordcount.SparkStreamin method), 30
__init__() (eskapadespark.links.spark_streaming_writer.SparkStreamingWr method), 31
__init__() (eskapadespark.links.spark_with_column.SparkWithColumn method), 32
__init__() (eskapadespark.links.sparkgeneralfuncprocessor.SparkGeneralFu method), 33
__init__() (eskapadespark.links.sparkhister.SparkHister method), 34
__init__() (eskapadespark.spark_manager.SparkManager method), 51

A

add_proc_func() (eskapadespark.links.HiveReader method), 46
assert_dataframe() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
assert_dataframe() (eskapadespark.links.SparkHistogrammarFiller method), 39

C

calc_asym() (in module eskapadespark.functions), 49
construct_empty_hist() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
construct_empty_hist() (eskapadespark.links.SparkHistogrammarFiller method), 39
create_session() (eskapadespark.spark_manager.SparkManager method), 51
create_spark_df() (in module eskapadespark.data_conversion), 46

D

DailySummary (class in eskapadespark.links), 44
DailySummary (class in eskapadespark.links.daily_summary), 18
df_schema() (in module eskapadespark.data_conversion), 47

E

eskapadespark (module), 52
eskapadespark.data_conversion (module), 46
eskapadespark.decorators (module), 48
eskapadespark.exceptions (module), 48
eskapadespark.functions (module), 49
eskapadespark.links (module), 34
eskapadespark.links.daily_summary (module), 18
eskapadespark.links.find_days_until_event (module), 19
eskapadespark.links.rdd_group_mapper (module), 20
eskapadespark.links.spark_configurator (module), 21
eskapadespark.links.spark_data_to_csv (module), 22
eskapadespark.links.spark_df_converter (module), 23
eskapadespark.links.spark_df_creator (module), 24
eskapadespark.links.spark_df_reader (module), 24
eskapadespark.links.spark_df_writer (module), 25
eskapadespark.links.spark_execute_query (module), 26
eskapadespark.links.spark_histogrammar_filler (module), 27
eskapadespark.links.spark_streaming_controller (module), 29
eskapadespark.links.spark_streaming_wordcount (module), 30
eskapadespark.links.spark_streaming_writer (module), 31
eskapadespark.links.spark_with_column (module), 31
eskapadespark.links.sparkgeneralfuncprocessor (module), 32
eskapadespark.links.sparkhister (module), 33
eskapadespark.resources (module), 51
eskapadespark.spark_manager (module), 51
eskapadespark.version (module), 52
execute() (eskapadespark.links.daily_summary.DailySummary method), 19
execute() (eskapadespark.links.DailySummary method), 44
execute() (eskapadespark.links.find_days_until_event.FindDaysUntilEvent method), 20
execute() (eskapadespark.links.FindDaysUntilEvent method), 45
execute() (eskapadespark.links.HiveReader method), 46
execute() (eskapadespark.links.HiveWriter method), 46
execute() (eskapadespark.links.rdd_group_mapper.RddGroupMapper method), 21
execute() (eskapadespark.links.RddGroupMapper method), 35
execute() (eskapadespark.links.spark_configurator.SparkConfigurator method), 22
execute() (eskapadespark.links.spark_data_to_csv.SparkDataToCsv method), 22
execute() (eskapadespark.links.spark_df_converter.SparkDfConverter method), 23
execute() (eskapadespark.links.spark_df_creator.SparkDfCreator method), 24
execute() (eskapadespark.links.spark_df_reader.SparkDfReader method), 25
execute() (eskapadespark.links.spark_df_writer.SparkDfWriter method), 26
execute() (eskapadespark.links.spark_execute_query.SparkExecuteQuery method), 26
execute() (eskapadespark.links.spark_streaming_controller.SparkStreaming method), 30
execute() (eskapadespark.links.spark_streaming_wordcount.SparkStreaming method), 30
execute() (eskapadespark.links.spark_streaming_writer.SparkStreamingWri method), 31
execute() (eskapadespark.links.spark_with_column.SparkWithColumn method), 32
execute() (eskapadespark.links.SparkConfigurator method), 35
execute() (eskapadespark.links.SparkDataToCsv method), 36
execute() (eskapadespark.links.SparkDfConverter method), 36
execute() (eskapadespark.links.SparkDfCreator method), 37
execute() (eskapadespark.links.SparkDfReader method),

37
execute() (eskapadespark.links.SparkDfWriter method), 38
execute() (eskapadespark.links.SparkExecuteQuery method), 38
execute() (eskapadespark.links.SparkGeneralFuncProcessor method), 43
execute() (eskapadespark.links.sparkgeneralfuncprocessor.SparkGeneralFuncProcessor method), 33
execute() (eskapadespark.links.SparkHister method), 43
execute() (eskapadespark.links.sparkhister.SparkHister method), 34
execute() (eskapadespark.links.SparkStreamingController method), 40
execute() (eskapadespark.links.SparkStreamingWordCount method), 41
execute() (eskapadespark.links.SparkStreamingWriter method), 41
execute() (eskapadespark.links.SparkWithColumn method), 42

F

fill_histogram() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
fill_histogram() (eskapadespark.links.SparkHistogrammarFiller method), 40
finalize() (eskapadespark.links.daily_summary.DailySummary method), 19
finalize() (eskapadespark.links.DailySummary method), 44
finalize() (eskapadespark.links.find_days_until_event.FindDaysUntilEvent method), 20
finalize() (eskapadespark.links.FindDaysUntilEvent method), 45
finalize() (eskapadespark.links.spark_streaming_controller.SparkStreamingController method), 30
finalize() (eskapadespark.links.spark_streaming_wordcount.SparkStreamingWordCount method), 30
finalize() (eskapadespark.links.spark_streaming_writer.SparkStreamingWriter method), 31
finalize() (eskapadespark.links.spark_with_column.SparkWithColumn method), 32
finalize() (eskapadespark.links.SparkHister method), 43
finalize() (eskapadespark.links.sparkhister.SparkHister method), 34
finalize() (eskapadespark.links.SparkStreamingController method), 40
finalize() (eskapadespark.links.SparkStreamingWordCount method), 41
finalize() (eskapadespark.links.SparkStreamingWriter method), 41
finalize() (eskapadespark.links.SparkWithColumn method), 42
FindDaysUntilEvent (class in eskapadespark.links), 44
FindDaysUntilEvent (class in eskapadespark.links.find_days_until_event), 19
finish() (eskapadespark.spark_manager.SparkManager method), 51

G

get_all_columns() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
get_all_columns() (eskapadespark.links.SparkHistogrammarFiller method), 40
get_data_type() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
get_data_type() (eskapadespark.links.SparkHistogrammarFiller method), 40
get_session() (eskapadespark.spark_manager.SparkManager method), 51

H

hgr_patch_histogram() (in module eskapadespark.links.spark_histogrammar_filler), 29
hgr_reset_quantity() (in module eskapadespark.links.spark_histogrammar_filler), 29
hive_table_from_df() (in module eskapadespark.data_conversion), 48
HiveReader (class in eskapadespark.links), 45
HiveWriter (class in eskapadespark.links), 46

I

initialize() (eskapadespark.links.daily_summary.DailySummary method), 19
initialize() (eskapadespark.links.DailySummary method), 44
initialize() (eskapadespark.links.find_days_until_event.FindDaysUntilEvent method), 20
initialize() (eskapadespark.links.FindDaysUntilEvent method), 45
initialize() (eskapadespark.links.HiveReader method), 46
initialize() (eskapadespark.links.HiveWriter method), 46
initialize() (eskapadespark.links.rdd_group_mapper.RddGroupMapper method), 21
initialize() (eskapadespark.links.RddGroupMapper method), 35
initialize() (eskapadespark.links.spark_configurator.SparkConfigurator method), 22
initialize() (eskapadespark.links.spark_data_to_csv.SparkDataToCsv method), 23
initialize() (eskapadespark.links.spark_df_converter.SparkDfConverter method), 24
initialize() (eskapadespark.links.spark_df_creator.SparkDfCreator method), 24

initialize() (eskapadespark.links.spark_df_reader.SparkDfReader method), 25
initialize() (eskapadespark.links.spark_df_writer.SparkDfWriter method), 26
initialize() (eskapadespark.links.spark_execute_query.SparkExecuteQuery method), 27
initialize() (eskapadespark.links.spark_streaming_controller.SparkStreamingController method), 30
initialize() (eskapadespark.links.spark_streaming_wordcount.SparkStreamingWordCount method), 30
initialize() (eskapadespark.links.spark_streaming_writer.SparkStreamingWriter method), 31
initialize() (eskapadespark.links.spark_with_column.SparkWithColumn method), 32
initialize() (eskapadespark.links.SparkConfigurator method), 35
initialize() (eskapadespark.links.SparkDataToCsv method), 36
initialize() (eskapadespark.links.SparkDfConverter method), 36
initialize() (eskapadespark.links.SparkDfCreator method), 37
initialize() (eskapadespark.links.SparkDfReader method), 37
initialize() (eskapadespark.links.SparkDfWriter method), 38
initialize() (eskapadespark.links.SparkExecuteQuery method), 38
initialize() (eskapadespark.links.SparkGeneralFuncProcessor method), 43
initialize() (eskapadespark.links.sparkgeneralfuncprocessor.SparkGeneralFuncProcessor method), 33
initialize() (eskapadespark.links.SparkHister method), 43
initialize() (eskapadespark.links.sparkhister.SparkHister method), 34
initialize() (eskapadespark.links.SparkStreamingController method), 41
initialize() (eskapadespark.links.SparkStreamingWordCount method), 41
initialize() (eskapadespark.links.SparkStreamingWriter method), 42
initialize() (eskapadespark.links.SparkWithColumn method), 42
is_inf() (in module eskapadespark.functions), 49
is_nan() (in module eskapadespark.functions), 50

M

MissingJayDeBeApiError, 48
MissingPy4jError, 49
MissingSparkError, 49

P

process_and_store() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 28
process_and_store() (eskapadespark.links.SparkHistogrammarFiller method), 40
process_columns() (eskapadespark.links.spark_histogrammar_filler.SparkHistogrammarFiller method), 29
process_columns() (eskapadespark.links.SparkHistogrammarFiller method), 40

R

RddGroupMapper (class in eskapadespark.links), 34
RddGroupMapper (class in eskapadespark.links.rdd_group_mapper), 20

S

spark_cls_reduce() (in module eskapadespark.decorators), 48
spark_query_func() (in module eskapadespark.functions), 50
spark_sql_func() (in module eskapadespark.functions), 50
spark_streaming_context (eskapadespark.spark_manager.SparkManager attribute), 51
SparkConfigurator (class in eskapadespark.links), 35
SparkConfigurator (class in eskapadespark.links.spark_configurator), 21
SparkDataToCsv (class in eskapadespark.links), 35
SparkDataToCsv (class in eskapadespark.links.spark_data_to_csv), 22
SparkDfConverter (class in eskapadespark.links), 36
SparkDfConverter (class in eskapadespark.links.spark_df_converter), 23
SparkDfCreator (class in eskapadespark.links), 36
SparkDfCreator (class in eskapadespark.links.spark_df_creator), 24
SparkDfReader (class in eskapadespark.links), 37
SparkDfReader (class in eskapadespark.links.spark_df_reader), 25
SparkDfWriter (class in eskapadespark.links), 37
SparkDfWriter (class in eskapadespark.links.spark_df_writer), 25
SparkExecuteQuery (class in eskapadespark.links), 38
SparkExecuteQuery (class in eskapadespark.links.spark_execute_query), 26
SparkGeneralFuncProcessor (class in eskapadespark.links), 42
SparkGeneralFuncProcessor (class in eskapadespark.links.sparkgeneralfuncprocessor), 32
SparkHister (class in eskapadespark.links), 43
SparkHister (class in eskapadespark.links.sparkhister), 34

SparkHistogrammarFiller (class in eskapadespark.links), 38
SparkHistogrammarFiller (class in eskapadespark.links.spark_histogrammar_filler), 27
SparkManager (class in eskapadespark.spark_manager), 51
SparkStreamingController (class in eskapadespark.links), 40
SparkStreamingController (class in eskapadespark.links.spark_streaming_controller), 29
SparkStreamingWordCount (class in eskapadespark.links), 41
SparkStreamingWordCount (class in eskapadespark.links.spark_streaming_wordcount), 30
SparkStreamingWriter (class in eskapadespark.links), 41
SparkStreamingWriter (class in eskapadespark.links.spark_streaming_writer), 31
SparkWithColumn (class in eskapadespark.links), 42
SparkWithColumn (class in eskapadespark.links.spark_with_column), 32

T

to_date_time() (in module eskapadespark.functions), 50
to_timestamp() (in module eskapadespark.functions), 50

U

unit_func() (in module eskapadespark.links.spark_histogrammar_filler), 29