Eskapade Documentation

KPMG Advanced Analytics
Big Data team

Nov 15, 2018

Contents

1 Documentation 3

2 Check it out 5

3 Quick run 7

4 Release notes 9

5 Contact and support 11

6 Contents 13
- 6.1 Introduction . . . 13
- 6.2 Installation . . . 14
- 6.3 Tutorials . . . 20
- 6.4 Command Line Arguments . . . 37
- 6.5 Package structure . . . 41
- 6.6 Release notes . . . 44
- 6.7 Developing and Contributing . . . 46
- 6.8 References . . . 47
- 6.9 API . . . 47
- 6.10 Appendices . . . 140
- 6.11 Indices and tables . . . 143

Python Module Index 145

- Version: 0.8
- Released: Aug 2018
- Web page: http://eskapade.kave.io
- Repository: http://github.com/kaveio/eskapade

Eskapade is a light-weight, python-based data analysis framework, meant for developing and modularizing all sorts of data analysis problems into reusable analysis components.

Eskapade uses a modular approach to analytics, meaning that you can use pre-made operations (called 'links') to build an analysis. This is implemented in a chain-link framework, where you define a 'Chain', consisting of a number of Links. These links are the fundamental building block of your analysis. For example, a data loading link and a data transformation link will frequently be found together in a pre-processing Chain.

Each chain has a specific purpose, for example: data quality checks of incoming data, pre-processing of data, booking and/or training of predictive algorithms, validation of these algorithms, and their evaluation. By using this work methodology, analysis links can be more easily reused in future data analysis projects.

Eskapade is analysis-library agnostic. It is used to set up typical data analysis problems from multiple packages, e.g.: scikit-learn, Spark MLlib, and ROOT. Likewise, Eskapade can use a manner of different data structures to handle data, such as: pandas DataFrames, numpy arrays, Spark DataFrames/RDDs, and more.

For example, Eskapade has been used as a self-learning framework for typical machine learning problems. Trained algorithms can predict real-time or batch data, these models can be evaluated over time, and Eskapade can bookkeep and retrain their algorithms.

CHAPTER 1

Documentation

The entire Eskapade documentation including tutorials can be found here.

CHAPTER 2

Check it out

Eskapade requires Python 3 and is pip friendly. To get started, simply do:

```
$ pip install Eskapade
```

or check out the code from out github repository:

```
$ git clone git@github.com:KaveIO/Eskapade.git
$ pip install -e Eskapade/
```

where in this example the code is installed in edit mode (option -e).

You can now use Eskapade in Python with:

```
import eskapade
```

Congratulations, you are now ready to use Eskapade!

CHAPTER 3

Quick run

To see the available Eskapade example, do:

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

E.g. you can now run:

```
$ eskapade_run $TUTDIR/esk101_helloworld.py
```

For all available Eskapade example macros, please see our tutorials section.

CHAPTER 4

Release notes

In version 0.8 of Eskapade (August 2018) the modules `root-analysis` and `spark-analysis` have been split off into separate packages called Eskapade-ROOT and Eskapade-Spark .

So the (core) Eskapade package no longer depends on ROOT and Spark, just on plain python packages. This make it much easier for people to try out the core functionality of Eskapade.

To install Eskapade-ROOT and Eskapade-Spark, do:

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

or check out the code from out github repository:

```
$ git clone git@github.com:KaveIO/Eskapade-ROOT.git eskapade-root
$ pip install -e eskapade-root/
$ git clone git@github.com:KaveIO/Eskapade-Spark.git eskapade-spark
$ pip install -e eskapade-spark/
```

where in this example the code is installed in edit mode (option -e).

You can now use these in Python with:

```python
import eskapadespark
import esroofit
```

See release notes for previous versions of Eskapade.

CHAPTER 5

Contact and support

- Issues & Ideas: https://github.com/kaveio/eskapade/issues
- Q&A 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 6

Contents

6.1 Introduction

Welcome to Eskapade! This is a short introduction of the package and why we built it. In the next sections we will go over the installation, a tutorial on how to run Eskapade properly, some examples use-cases, and how to develop analysis code and run it in Jupyter and PyCharm.

6.1.1 What is Eskapade?

Eskapade is an abbreviation for: 'Enterprise Solution KPMG Advanced Predictive Analytics Decision Engine'. It is a light-weight Python framework to help make your data analytics modular. This results in faster roll-out of analytics solutions in your business and less overhead when taking multiple analyses in production. In particular, it is intended for building data processing pipe lines and using implemented machine learning models in a production environment.

6.1.2 Why did we build this?

We found that the implementation phase of a data analytics solution at clients - a careful process - is often completely different from building the solution itself - which proceeds through explorative iterations. Therefore we made this analysis framework that makes it easier to set up a data analysis, while simultaneously making it easier to put it into production.

Next to that, it makes analyses modular, which has a lot of advantages. It is easier to work with multiple people on the same project, because the contributions are divided in clear blocks. Re-use of code becomes more straightforward, as old code is already put in a block with a clear purpose. Lastly, it gives you a universal basis for all your analyses, which can both be used across a company, or for different clients.

More about the purpose can be read at the general readme.

6.2 Installation

Let's get Eskapade up and running! In order to make this as easy as possible, we provide both a Docker image and a virtual machine where everything you need is installed and working properly. Alternatively, you can download the repository and run it on your own machine.

- See *Eskapade on your own machine* for the local installation requirements.
- See *Eskapade with Docker* to get started with Docker.
- See *Eskapade on a virtual machine* to get started with Vagrant.

This manual is written for Linux systems, but Eskapade also runs fine on macOS systems.

6.2.1 Eskapade on your own machine

Eskapade can be installed as any other Python package with `easy_install` or `pip`. To get started, simply do:

```
$ pip install Eskapade
```

We have verified that this works on Linux 16.04 and MacOS based machines.

Or check out the code from out github repository:

```
$ git clone git@github.com:KaveIO/Eskapade.git
$ pip install -e Eskapade/
```

where the code is installed in editable mode (option -e).

You can now use Eskapade in Python with:

```
import eskapade
```

Congratulations, you are now ready to use Eskapade!

See the other parts of the documentation for specific usage.

6.2.2 Eskapade with Docker

Consistent environments for Eskapade development and use can be created with docker containers. Containers are created and managed by Docker. An Eskapade container contains a pre-installed Eskapade setup and runs out-of-the-box. It is possible to mount your customised Eskapade code for development inside the container.

The instructions below show how one can use a locally checked-out version of Eskapade in combination with this Docker image. This combination is a very convenient way of actively working with and/or developing code in Eskapade.

By default, Eskapade code is executed as `root` user in the container. It is possible, however, to run with reduced user privileges inside containers through user mapping with the Docker host, as decribed in the last section.

Required software

Docker installation instructions can be found here: https://docs.docker.com/install/.

Eskapade source code is optional (for development purposes), you can check it out locally with the command:

```
git clone git@github.com:KaveIO/Eskapade.git eskapade
```

Getting Eskapade docker images

From DockerHub

The official Eskapade docker image is provided on DockerHub.

```
$  docker pull kave/eskapade-env:0.8
```

This will download the `kave/eskapade-env:0.8` image locally. Downloading this docker image can take a minute or two.

Building from scratch

To build the docker image from scratch using the Eskapade source code, do:

```
$  cd eskapade/docker/eskapade-env && sh create_docker.sh
```

This will produce the `kave/eskapade-env:0.8` image locally.

Spinning up docker containers

Out-of-the-box

From this image, containers with the Eskapade environment set up, can be run out-of-the-box:

```
$  docker run -p 8888:8888 -it kave/eskapade-env:0.8
```

Where port 8888 is forwarded to the docker host to make Jupyter notebook available (below).

E.g. one can now do:

```
cd /opt/eskapade
eskapade_run --help
```

and run any Eskapade code. See the Tutorial section for examples.

Exit the (docker) bash shell with:

```
exit
```

See section *After you exit Docker* (below) for cleaning up obsolete docker processes.

Mounting source code

```
$  docker run -v <ESKAPADE>:/opt/eskapade -p 8888:8888 -it kave/eskapade-env:0.8
```

Where `<ESKAPADE>` specifies the path of the Eskapade source code on the docker host, and where `/opt/eskapade` is the location of the Eskapade source code inside the container.

NOTE: in case you mount a clean installation of the Eskapade source code, you have to (re-)build the libraries by executing:

```
$ pip install -e /opt/eskapade
```

Running as non-root user

For increased security in a production environment, it is recommended to run Eskapade code inside the container as non-root user. The `Dockerfile` in the `eskapade-user` directory provides an additional user-mapping layer to the `eskapade-env` image: it creates a `esdev` user that has its own virtual Python environment with Eskapade installed. The mapping of user id's between Docker host and container ensure that proper permissions are propogated when writing/reading to the mounted volume with Eskapade code.

To obtain a centrally produced Eskapade image, use:

```
$ docker pull kave/eskapade-usr:0.8
```

Or build the Eskapade docker image with `esdev` user installation, from scratch:

```
$  cd docker/eskapade-usr && docker build -t kave/eskapade-usr:0.8 .
```

This will produce the `kave/eskapade-usr:0.8` image.

From this image, containers with the Eskapade environment set up, can be run out-of-the-box:

```
$ docker run -e HOST_USER_ID=$(id -u) -e HOST_USER_GID=$(id -g) -p 8888:8888 -it kave/
↪eskapade-usr:0.8
```

The first time you run this command it will likely take some time. The `HOST_USER_ID` and `HOST_USER_GID` environment variables are used to dynamically map user- and group id's between the host and Docker container, ensuring proper read/write permissions.

Remapping the user id

To prevent the remapping of user and group id from happening the next time you boot up the image, open another shell:

```
$ docker ps
```

Copy the top CONTAINER-ID string, matching the running instance of the `kave/eskapade-usr:0.8` image, and then paste it:

```
$ docker commit CONTAINER-ID kave/eskapade-usr:0.8
```

Next time when you run:

```
$ docker run -e HOST_USER_ID=$(id -u) -e HOST_USER_GID=$(id -g) -p 8888:8888 -it kave/
↪eskapade-usr:0.8
```

the remapping of user and group id should no longer happen.

Mounting source code

Containers with the user-specific Eskapade environment setup can be run out-of-the-box, and with your own mounted (customised) source code, using:

```
$  docker run -e HOST_USER_ID=$(id -u) -e HOST_USER_GID=$(id -g) -v <ESKAPADE>:/home/
↪esdev/eskapade -p 8888:8888 -it kave/eskapade-usr:0.8
```

Where `<ESKAPADE>` specifies the path of the Eskapade source code.

NOTE: in case you mount a clean installation of the Eskapade source code, you have to (re-)build the libraries by executing:

```
$ pip install -e /home/esdev/eskapade
```

This combination is a great way of using and developing Eskapade code.

Consider adding a permanent alias to your local `~/.bashrc` or `~/.bash_profile` file:

```
alias eskapade_docker='docker run -e HOST_USER_ID=$(id -u) -e HOST_USER_GID=$(id -g) -
↪v <ESKAPADE>:/home/esdev/eskapade -p 8888:8888 -it kave/eskapade-usr:0.8'
```

So the next time, in a fresh shell, you can simply run the command `eskapade_docker`.

Starting Jupyter notebook

To run the Jupyter notebook on port 8888 from the docker environment:

```
cd /opt/eskapade
jupy &
```

And press enter twice to return to the shell prompt.

The command `jupy &` starts up Jupyter notebook in the background on port 8888 and pipes the output to the log file `nohup.out`.

In your local browser then go to address:

```
localhost:8888/
```

And you will see the familiar Jupyter environment. In case you get asked for a password, take a look at the tail end of the file `nohup.out`, where you will see the exact url address that you need to go to.

E.g. you can now do `import eskapade` (shift-enter) to get access to the Eskapade library.

Be sure to run `jupy &` from a directory that is mounted in the docker container, such as `/opt/eskapade`. In this way any notebook(s) you create are kept after you exit the docker run.

After you exit Docker

Every time you want to have a clean Docker environment, run the following commands:

```
# --- 1. remove all exited docker processes
docker ps -a | grep Exited | awk '{print "docker stop "$1 "; docker rm "$1}' | sh

# --- 2. remove all failed docker image builts
docker images | grep "<none>" | awk '{print "docker rmi "$3}' | sh
```

(continues on next page)

(continued from previous page)

```
# --- 3. remove dangling volume mounts
docker volume ls -qf dangling=true | awk '{print "docker volume rm "$1}' | sh
```

To automate this, we advise you put these commands in an executable `docker_cleanup.sh` script.

6.2.3 Eskapade on a virtual machine

Consistent environments for Eskapade development and use are created with virtual machines. Machines are created and managed by Vagrant, with VirtualBox as a provider.

Getting started

Required software

To build and run Eskapade boxes, Vagrant and VirtualBox are required. Vagrant can be downloaded at https://www.vagrantup.com/downloads.html. For Ubuntu (and other Debian-based systems), download the Vagrant Debian package, move to where you want to install this and run this command to install:

```
dpkg -i vagrant_<VERSION>_<ARCHITECTURE>.deb
```

where the version and architecture are in the file name.

VirtualBox can be downloaded and installed by following the instructions on https://www.virtualbox.org/wiki/Downloads. Make sure you install the latest version from the VirtualBox repository instead of an older version from the repositories of your system.

Repository

The repository, which contains the code to build your virtual environment, is hosted on github. Clone it to your machine with:

```
$ git clone git@github.com:KaveIO/Eskapade.git
```

All code below is executed in the root of this repository, unless otherwise stated.

Virtual environment

The environment box contains Ubuntu and the KAVE Toolbox, including Anaconda, Spark, and ROOT.

You can either build this box yourself, or download the image. In order to build it yourself, run the following commands. It might take a while to build (up to two hours!)

```
cd vagrant/dev
vagrant up
```

Alternatively, in order to download and import its image, do:

```
cd vagrant/dev_image
vagrant box add eskapade-dev.json
vagrant up
```

Eskapade users and developers log in as the user `esdev` on `localhost`, by default on port 2222:

```
ssh -p 2222 esdev@localhost
```

This user can log in with the password `esdev`.

By default the local directory `/path/to/your/local/eskapade` is mounted (containing your local Eskapade repository) under `/opt/eskapade` in your vagrant machine. You can now edit the files in this directory, either locally or in the (vagrant) bash shell, and any updates to these files will be kept after exiting the virtual machine.

You are now ready to use Eskapade!

The next time…

Simply do:

```
cd vagrant/dev
vagrant up
```

or:

```
cd vagrant/dev_image
vagrant up
```

depending on whether you built vagrant yourself or downloaded the image.

Then you can access it via ssh (password `esdev`):

```
ssh -p 2222 esdev@localhost
```

Easy log-in

To make logging in easier, the key pair `vagrant/dev/ssh/esdev_id_rsa.pub`, `vagrant/dev/ssh/esdev_id_rsa` can be used, and an example SSH configuration is provided in `vagrant/dev/ssh/config`. Put these files in your `~/.ssh/`:

```
cp vagrant/dev/ssh/* ~/.ssh/
```

You can then log in using the command:

```
ssh esdevbox
```

Vagrant boxes

Boxes are built and started with the command `vagrant up` in the directory of the `Vagrantfile` describing the box. A box can be restarted by executing `vagrant reload`. The virtual machines are administered by the `vagrant` user, which logs in by running `vagrant ssh` in the directory of the `Vagrantfile`. The `vagrant` user has root access to the system by password-less `sudo`.

Starting Jupyter notebook

To run the Jupyter notebook on port 8888 from the vagrant machine:

```
cd /opt/eskapade
jupy &
```

And press enter twice to return to the shell prompt.

The command `jupy &` starts up Jupyter notebook in the background on port 8888 and pipes the output to the log file `nohup.out`.

In your local browser then go to address:

> localhost:8888/

And you will see the familiar Jupyter environment.

E.g. you can now do `import eskapade` (shift-enter) to get access to the Eskapade library.

Be sure to run `jupy &` from a directory that is mounted in the vagrant machine, such as `/opt/eskapade`. In this way any notebook(s) you create are kept after you exit the docker run.

Requirements

Eskapade requires Python 3 and some libraries, which can be found in *setup.py* at the root of the repository.

Eskapade can be installed as any other Python package with `easy_install` or `pip`:

```
$ pip install /path/to/eskapade
```

Alternatively, consider installing KaveToolbox version 3.6 or higher. To install the released version:

```
$ yum -y install wget curl tar zip unzip gzip python
$ wget http://repos:kaverepos@repos.kave.io/noarch/KaveToolbox/3.6-Beta/kavetoolbox-
↪installer-3.6-Beta.sh
$ sudo bash kavetoolbox-installer-3.6-Beta.sh [--quiet]
```

(–quiet is for a quieter install, remove the brackets!)

If anaconda is already installed in your machine, consider creating a conda virtual environment with Python 3.6 to install all the requirements and Eskapade itself to avoid collisions:

```
$ conda create -n eskapade_env36 python=3.6 anaconda
```

Then you can activate it as follows:

```
$ source activate eskapade_env36
```

More information about conda virtual environments can be found here

6.2.4 Installing Eskapade on macOS

To install eskapade on macOS, see our macOS appendix.

6.3 Tutorials

This section contains materials on how to use Eskapade. There are additional side notes on how certain aspects work and where to find parts of the code. For more in depth explanations on the functionality of the code-base, try the API docs.

All command examples below can be run from any directory with write access.

6.3.1 Running your first macro

After successfully installing Eskapade, it is now time to run your very first macro, the classic code example: Hello World!

For ease of use, let’s make a shortcut to the directory containing the Eskapade tutorials:

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

Hello World!

If you just want to run it plain and simple, go to the root of the repository and run the following:

```
$ eskapade_run $TUTDIR/esk101_helloworld.py
```

This will run the macro that prints out Hello World. There is a lot of output, but try to find back these lines (or similar):

```
2017-11-13T12:37:07.473512+00:00 [eskapade.core_ops.links.hello_world.HelloWorld
↪#INFO] Hello World
2017-11-13T12:37:07.473512+00:00 [eskapade.core_ops.links.hello_world.HelloWorld
↪#INFO] Hello World
```

Congratulations, you have just successfully run Eskapade!

Internal workings

To see what is actually happening under the hood, go ahead and open up `tutorials/esk101_helloworld.py`. The macro is like a recipe and it contains all of your analysis. It has all the ‘high level’ operations that are to be executed by Eskapade.

When we go into this macro we find the following piece of code:

```
hello = Chain(name='Hello')
link = core_ops.HelloWorld(name='HelloWorld')
link.logger.log_level = LogLevel.DEBUG
link.repeat = settings['n_repeat']
hello.add(link)
```

Which is the code that does the actual analysis (in this case, print out the statement). In this case `link` is an instance of the class HelloWorld, which itself is a Link. The Link class is the fundamental building block in Eskapade that contains our analysis steps. The code for HelloWorld can be found at:

```
$ less python/eskapade/core_ops/links/hello_world.py
```

Looking into this class in particular, in the code we find in the `execute()` function:

```
self.logger.info('Hello {hello}', hello=self.hello)
```

where `self.hello` is a parameter set in the `__init__` of the class. This setting can be overwritten as can be seen below. For example, we can make another link, `link2` and change the default `self.hello` into something else.

```
link2 = core_ops.HelloWorld(name='Hello2')
link2.hello = 'Lionel Richie'
ch.add(link2)
```

Rerunning results in us greeting the famous singer/songwriter.

There are many ways to run your macro and control the flow of your analysis. You can read more on this in the *Short introduction to the Framework* subsection below.

6.3.2 Tutorial 1: transforming data

Now that we know the basics of Eskapade we can go on to more advanced macros, containing an actual analysis.

Before we get started, we have to fetch some data, on your command line, type:

```
$ wget https://s3-eu-west-1.amazonaws.com/kpmg-eskapade-share/data/LAozone.data
```

To run the macro type on your CLI:

```
$ eskapade_run $TUTDIR/tutorial_1.py
```

If you want to add command line arguments, for example to change the output logging level, read the page on command line arguments.

When looking at the output in the terminal we read something like the following:

```
2017-11-13T13:37:07.473512+00:00 [eskapade.core.execution#INFO] *                   ␣
↪Welcome to Eskapade!                        *
...
2017-11-13T13:37:08.085577+00:00 [eskapade.core.process_manager.ProcessManager#INFO]␣
↪Number of registered chains: 2
...
2017-11-13T13:37:11.316414+00:00 [eskapade.core.execution#INFO] *                   ␣
↪Leaving Eskapade. Bye!                      *
```

There is a lot more output than these lines (tens or hundred of lines depending on the log level). Eskapade has run the code from each link, and at the top of the output in your terminal you can see a summary.

When you look at the output in the terminal you can see that the macro contains two chains and a few Link are contained in these chains. Note that chain 2 is empty at this moment. In the code of the macro we see that in the first chain that data is loaded first and then a transformation is applied to this data.

Before we are going to change the code in the macro, there will be a short introduction to the framework.

Short introduction to the Framework

At this point we will not go into the underlying structure of the code that is underneath the macro, but later in this tutorial we will. For now we will take a look in the macro. So open `$TUTDIR/tutorial_1.py` in your favorite editor. We notice the structure: first imports, then defining all the settings, and finally the actual analysis: Chains and Links.

A chain is instantiated as follows:

```
data = Chain('Data')
```

and registered automatically with the ProcessManager. The ProcessManager is the main event processing loop and is responsible for processing the Chains and Links.

Next a Pandas data frame converter Link is initialized and its properties are set, and finally added to the data chain:

```
reader = analysis.ReadToDf(name='Read_LA_ozone', path='LAozone.data', reader=pd.read_
↪csv, key='data')
data.add(reader)
```

This means the Link is added to the chain and when Eskapade runs, it will execute the code in the Link.

Now that we know how the framework runs the code on a higher level, we will continue with the macro.

In the macro notice that under the second chain some code has been commented out. Uncomment the code and run the macro again with:

```
$ eskapade_run $TUTDIR/tutorial_1.py
```

And notice that it takes a bit longer to run, and the output is longer, since it now executes the Link in chain 2. This Link takes the data from chain 1 and makes plots of the data in the data set and saves it to your disk. Go to this path and open one of the pdfs found there:

```
$ results/Tutorial_1/data/v0/report/
```

The pdfs give an overview of all numerical variables in the data in histogram form. The binning, plotting and saving of this data is all done by the chain we just uncommented. If you want to take a look at how the Link works, it can be found in:

```
$ python/eskapade/visualization/links/df_summary.py
```

But for now, we will skip the underlying functionality of the links.

Let's do an exercise. Going back to the first link, we notice that the transformations that are executed are defined in `conv_funcs` passed to the link. We want to include in the plot the wind speed in km/h. There is already a part of the code available in the `conv_funcs` and the functions `comp_date` and `mi_to_km`. Use these functions as examples to write a function that converts the wind speed.

Add this to the transformation by adding your own code. Once this works you can also try to add the temperature in degrees Celsius.

Making a Link

Now we are going to add a new link that we create! To make a new link type the following:

```
$ eskapade_generate_link --dir python/eskapade/analysis/links YourLink
```

The command will make a link object named `YourLink` in the path specified in the first argument. The link we wish to add will do some textual transformation, so name it accordingly. And be sure to follow the instructions given by the command!

The command creates the skeleton file:

```
$ python/eskapade/analysis/links/yourlink.py
```

This skeleton file can be modified with your custom editor and then be imported and called inside a macro with `analysis.YourLink()`. Notice that the name of the class is CamelCase and that the name of the file is lowercase to conform to coding guidelines.

Now open up the link in your editor. In the `execute` function of the Link, we see that a DataStore is called. This is the central in-memory object in which all data is saved. DataStore inherits from a dict, so by calling the right key we can get objects. Call:

```
df = ds['data']
```

to get the DataFrame that includes the latest transformations.

Now we are going to make a completely different transformation in the Link and apply it to the object in the DataStore. We want to add a column to the data that states how humid it is. When column 'humidity' is less than 50 it is 'dry', otherwise it is 'humid'. You will have to use some pandas functionality or perhaps something else if you prefer. Save the new column back into the DataFrame and then put the DataFrame in the DataStore under the key 'data_new'.

We are going to let our plot functionality loose on this DataFrame once more, to see what happens to our generated textual data. It can not be plotted. In the future this functionality will be available for most data types.

Now run the entire macro with the new code and compile the output .tex file. This can be done on the command line with

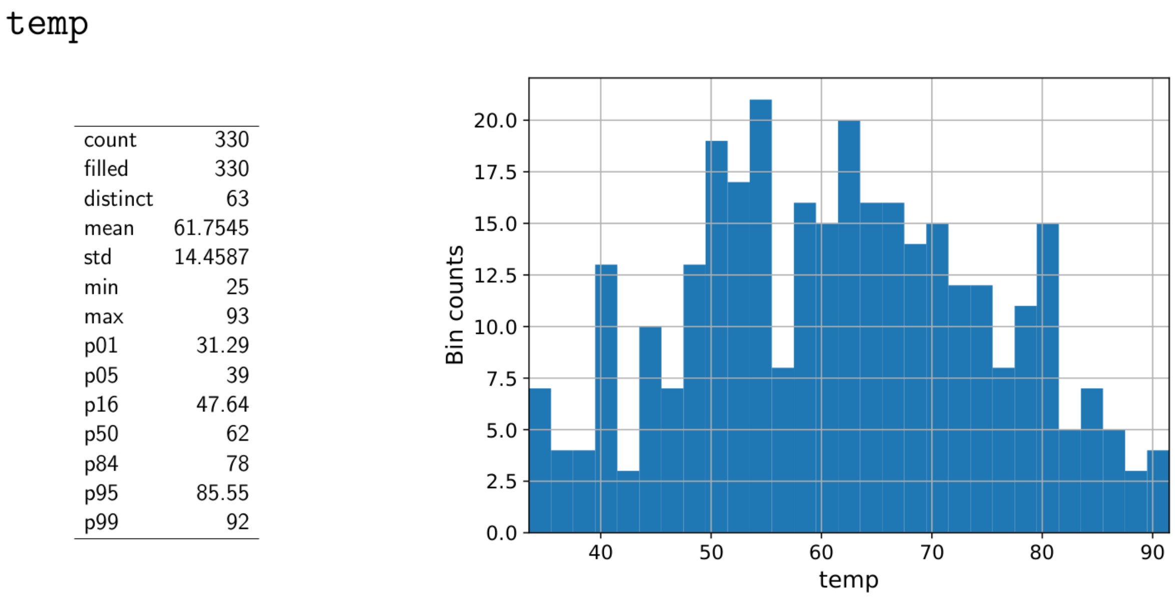
```
$ cd results/Tutorial_1/data/v0/report/
$ pdflatex report.tex
```

If you have pdflatex installed on your machine.

Note: If you don't have pdflatex installed on your machine you can install it by executing the following command: .. code-block:: bash

> $ yum install texlive-latex-recommended

Now take a look at the output pdf. The final output should look something like this:



temp

count	330
filled	330
distinct	63
mean	61.7545
std	14.4587
min	25
max	93
p01	31.29
p05	39
p16	47.64
p50	62
p84	78
p95	85.55
p99	92

Your plot should be quite similar (though it might be different in its make up.)

In summary, the work method of Eskapade is to run chains of custom code chunks (links). Each chain should have a specific purpose, for example pre-processing incoming data, booking and/or training predictive algorithms, validating these predictive algorithms, evaluating the algorithms.

By using this work method, links can be easily reused in future projects. Some links are provided by default. For example, links used to load data from a json file, book predictive algorithms, predict the training and test data set and produce evaluation metrics. If you want to use your own predictive model just go ahead and add your own links!

6.3.3 Tutorial 2: macro from basic links

In this tutorial we are going to build a macro using existing Links. We start by using templates to make a new macro. The command

```
$ eskapade_generate_macro --dir python/eskapade/tutorials tutorial_2
```

makes a new macro from a template macro. When we open the macro we find a lot of options that we can use. For now we will actually not use them, but if you want to learn more about them, read the Examples section below.

First we will add new chains to the macro. These are the higher level building blocks that can be controlled when starting a run of the macro. At the bottom of the macro we find a commented out Link, the classic Hello World link. You can uncomment it and run the macro if you like, but for now we are going to use the code to make a few chains.

So use the code and add 3 chains with different names:

```
ch = Chain('CHAINNAME')
```

When naming chains, remember that the output of Eskapade will print per chain-link combination the logs that are defined in the Links. So name the chains appropriately, so when you run the macro the logging actually makes sense.

This tutorial will be quite straight-forward, it has 3 short steps, which is why we made 3 chains.

1. In the first chain: Read a data file of your choosing into Eskapade using the pandas links in the analysis subpackage.
2. In the second chain: Copy the DataFrame you created in the DataStore using the core_ops subpackage.
3. In the third chain: Delete the entire DataStore using a Link in the core_ops subpackage.

To find the right Links you have to go through the Eskapade documentation (or code!), and to find within its subpackages the proper Links you have to understand the package structure. Every package is specific for a certain task, such as analysis, core tasks (like the `ProcessManager`), or data quality. Every subpackage contains links in its `links/` subdirectory. See for example the subpackages `core_ops`, `analysis` or `visualization`.

In *All available examples* we give some tips to find the right Links your analysis, and how to configure them properly.

6.3.4 Tutorial 3: Jupyter notebook

This section contains materials on how to use Eskapade in Jupyter Notebooks. There are additional side notes on how certain aspects work and where to find parts of the code. For more in depth explanations, try the API-docs.

Next we will demonstrate how Eskapade can be run and debugged interactively from within a Jupyter notebook.

An Eskapade notebook

To run Eskapade use the `eskapade_generate_notebook` command to create a template notebook. For example:

```
$ eskapade_generate_notebook --dir ./ notebook_name
```

The minimal code you need to run a notebook is the following:

```
from eskapade import process_manager, resources, ConfigObject, DataStore
from eskapade.core import execution, persistence
from eskapade.logger import LogLevel

# --- basic config
settings = process_manager.service(ConfigObject)
settings['macro'] = resources.tutorial('tutorial_1.py')
settings['version'] = 0
settings['logLevel'] = LogLevel.DEBUG

# --- optional running parameters
# settings['beginWithChain'] = 'startChain'
# settings['endWithChain'] = 'endChain'
# settings['resultsDir'] = 'resultsdir'
settings['storeResultsEachChain'] = True

# --- other global flags (just some examples)
# settings['set_mongo'] = False
# settings['set_training'] = False

# --- run eskapade!
execution.eskapade_run(settings)

# --- To rerun eskapade, clear the memory state first!
# execution.reset_eskapade()
```

Make sure to fill out all the necessary parameters for it to run. The macro has to be set obviously, but not all settings in this example are needed to be set to a value. The function `execution.eskapade_run(settings)` runs Eskapade with the settings you specified.

To inspect the state of the Eskapade objects (datastore and configurations) after the various chains see the command line examples below. .. note:

```
Inspecting intermediate states requires Eskapade to be run with the option␣
↪storeResultsEachChain
(command line: ``-w``) on.
```

```
# --- example inspecting the data store after the preprocessing chain
ds = DataStore.import_from_file('./results/Tutorial_1/proc_service_data/v0/_Summary/
↪eskapade.core.process_services.DataStore.pkl')
ds.keys()
ds.Print()
ds['data'].head()

# --- example showing Eskapade settings
settings = ConfigObject.import_from_file('./results/Tutorial_1/proc_service_data/v0/_
↪Summary/eskapade.core.process_services.ConfigObject.pkl')
settings.Print()
```

The `import_from_file` function imports a pickle file that was written out by Eskapade, containing the DataStore. This can be used to start from an intermediate state of your Eskapade. For example, you do some operations on your DataStore and then save it. At a later time you load this saved DataStore and continue from there.

Running in a notebook

In this tutorial we will make a notebook and run the macro from tutorial 1. This macro shows the basics of Eskapade. Once we have Eskapade running in a terminal, we can run it also in Jupyter. Make sure you have properly installed Jupyter.

We start by making a notebook:

```
$ eskapade_generate_notebook tutorial_3_notebook
```

This will create a notebook in the current directory with the name `tutorial_3_notebook` running macro `tutorial_1.py`. You can set a destination directory by specifying the command argument `--dir`. Now open Jupyter and take a look at the notebook.

```
$ jupyter notebook
```

Try to run the notebook. You might get an error if the notebook can not find the data for the data reader. Unless you luckily are in the right folder. By default, `tutorial_1.py` looks for the data file `LAozone.data` in the working directory. Use:

```
!pwd
```

In Jupyter to find which path you are working on, and put the data to the path. Or change the load path in the macro to the proper one. But in the end it depends on your setup.

Intermezzo: you can run bash commands in Jupyter by prepending the command with a !

Now run the cells in the notebook and check if the macro runs properly. The output be something like:

```
2017-02-14 14:04:55,506 DEBUG [link/execute_link]: Now executing link 'LA ozone data'
2017-02-14 14:04:55,506 DEBUG [readtodf/execute]: reading datasets from files ["../
↪data/LAozone.data"]
2017-02-14 14:04:55,507 DEBUG [readtodf/pandasReader]: using Pandas reader "<function␣
↪_make_parser_function.<locals>.parser_f at 0x7faaac7f4d08>"
2017-02-14 14:04:55,509 DEBUG [link/execute_link]: Done executing link 'LA ozone data'
2017-02-14 14:04:55,510 DEBUG [link/execute_link]: Now executing link 'Transform'
2017-02-14 14:04:55,511 DEBUG [applyfunctodataframe/execute]: Applying function
↪<function <lambda> at 0x7faa8ba2e158>
2017-02-14 14:04:55,512 DEBUG [applyfunctodataframe/execute]: Applying function
↪<function <lambda> at 0x7faa8ba95f28>
2017-02-14 14:04:55,515 DEBUG [link/execute_link]: Done executing link 'Transform'
2017-02-14 14:04:55,516 DEBUG [chain/execute]: Done executing chain 'Data'
2017-02-14 14:04:55,516 DEBUG [chain/finalize]: Now finalizing chain 'Data'
2017-02-14 14:04:55,517 DEBUG [link/finalize_link]: Now finalizing link 'LA ozone data
↪'
2017-02-14 14:04:55,518 DEBUG [link/finalize_link]: Done finalizing link 'LA ozone␣
↪data'
2017-02-14 14:04:55,518 DEBUG [link/finalize_link]: Now finalizing link 'Transform'
2017-02-14 14:04:55,519 DEBUG [link/finalize_link]: Done finalizing link 'Transform'
2017-02-14 14:04:55,519 DEBUG [chain/finalize]: Done finalizing chain 'Data'
```

with a lot more text surrounding this output. Now try to run the macro again. The run should fail, and you get the following error:

```
KeyError: Processor "<Chain name=Data parent=<... ProcessManager ...> id=...>"␣
↪already exists!'
```

This is because the ProcessManager is a singleton. This means there is only one of this in memory allowed, and since the Jupyter python kernel was still running the object still existed and running the macro gave an error. The macro tried to add a chain, but it already exists in the ProcessManager. Therefore the final line in the notebook template has to be ran every time you want to rerun Eskapade. So run this line:

```
execution.reset_eskapade()
```

And try to rerun the notebook without restarting the kernel.

```
execution.eskapade_run(settings)
```

If one wants to call the objects used in the run, `execute` contains them. For example calling

```
ds = process_manager.service(DataStore)
```

is the DataStore, and similarly the other ‘master’ objects can be called. Resetting will clear the process manager singleton from memory, and now the macro can be rerun without any errors.

Note: restarting the Jupyter kernel also works, but might take more time because you have to re-execute all of the necessary code.

Reading data from a pickle

Continuing with the notebook we are going to load a pickle file that is automatically written away when the engine runs. First we must locate the folder where it is saved. By default this is in:

```
./results/$MACRO/proc_service_data/v$VERSION/latest/eskapade.core.process_services.
↪DataStore.pkl'
```

Where `$MACRO` is the macro name you specified in the settings, `$VERSION` is the version you specified and `latest` refers to the last chain you wrote to disk. By default, the version is `0` and the name is `v0` and the chain is the last chain of your macro.

You can write a specific chain with the command line arguments, otherwise use the default, the last chain of the macro.

Now we are going to load the pickle from tutorial_1.

So make a new cell in Jupyter and add:

```
from eskapade import DataStore
```

to import the DataStore module. Now to import the actual pickle and convert it back to the DataStore do:

```
ds = DataStore.import_from_file('./results/Tutorial_1/proc_service_data/v0/latest/
↪eskapade.core.process_services.DataStore.pkl')
```

to open the saved DataStore into variable `ds`. Now we can call the keys of the DataStore with

```
ds.Print()
```

We see there are two keys: `data` and `transformed_data`. Call one of them and see what is in there. You will find of course the pandas DataFrames that we used in the tutorial. Now you can use them in the notebook environment and directly interact with the objects without running the entirety of Eskapade.

Similarly you can open old ConfigObject objects if they are available. By importing and calling:

```
from eskapade import ConfigObject
settings = ConfigObject.import_from_file('./results/Tutorial_1/proc_service_data/v0/
↪latest/eskapade.core.process_services.ConfigObject.pkl')
```

one can import the saved singleton at the path. The singleton can be any of the above mentioned stores/objects. Finally, by default there are soft-links in the results directory at `results/$MACRO/proc_service_data/$VERSION/latest/` that point to the pickles of the associated objects from the last chain in the macro.

6.3.5 Tutorial 4: creating a new analysis project

Now that we have covered how to make a link, macro, and notebook we can create a new analysis project. To generate a new project type the following:

```
$ eskapade_bootstrap --project_root_dir ./yourproject -m yourmacro -l YourLink --n␣
↪yournotebook yourpackage
```

The script will create a Python package called `yourpackage` in the path specified in the `--project_root_dir` argument. The arguments `-m`, `-l`, and `-n` are optional, if not specified the default values are used.

The generated project has the following structure:

```
|-yourproject
   |-yourpackage
      |-links
         |-__init__.py
         |-yourlink.py
      |-__init__.py
      |-yourmacro.py
      |-yournotebook.ipynb
   |-setup.py
```

The project contains a link module called `yourlink.py` under `links` directory, a macro `yourmacro.py`, and a Jupyter notebook `yournotebook.ipynb` with required dependencies. To add more of each to the project you can use the commands `generate_link`, `generate_macro`, and `generate_notebook` like it was done before.

The script also generates `setup.py` file and the package can be installed as any other pip package.

Let's try to debug the project interactively within a Jupyter notebook. First, go to your project directory and install the package in an editable mode:

```
$ cd yourproject
$ pip install -e .
```

As you can see in the output, installation checks if `eskapade` and its requirements are installed.

If the installation was successful, run Jupyter and open `yournotebook.ipynb` in `yourpackage` directory:

```
$ jupyter notebook
```

As you see in the code the notebook runs `yourmacro.py`:

```
settings['macro'] = '<...>/yourproject/yourpackage/yourmacro.py'
```

Now run the cells in the notebook and check if the macro runs properly.

6.3.6 All available examples

To see the available Eskapade example, do:

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

Many Eskapade example macros exist in the tutorials directory. The numbering of the example macros follows the package structure:

- `esk100+`: basic macros describing the chains, links, and datastore functionality of Eskapade.
- `esk200+`: macros describing links to do basic processing of pandas dataframes.
- `esk300+`: visualization macros for making histograms, plots and reports of datasets.
- `esk500+`: macros for doing data quality assessment and cleaning.

The Eskapade macros are briefly described below. They explain the basic architecture of Eskapade, i.e. how the chains, links, datastore, and process manager interact.

Hopefully you now have enough knowledge to run and explore Eskapade by yourself. You are encouraged to run all examples to see what Eskapade can do for you!

Example esk101: Hello World!

Macro 101 runs the Hello World Link. It runs the Link twice using a repeat kwarg, showing how to use kwargs in Links.

```
$ eskapade_run $TUTDIR/esk101_helloworld.py
```

Example esk102: Multiple chains

Macro 102 uses multiple chains to print different kinds of output from one Link. This link is initialized multiple times with different kwargs and names. There are if-statements in the macro to control the usage of the chains.

```
$ eskapade_run $TUTDIR/esk102_multiple_chains.py
```

Example esk103: Print the DataStore

Macro 103 has some objects in the DataStore. The contents of the DataStore are printed in the standard output.

```
$ eskapade_run $TUTDIR/esk103_printdatastore.py
```

Example esk104: Basic DataStore operations

Macro 104 adds some objects from a dictionary to the DataStore and then moves or deletes some of the items. Next it adds more items and prints some of the objects.

```
$ eskapade_run $TUTDIR/esk104_basic_datastore_operations.py
```

Example esk105: DataStore Pickling

Macro 105 has 3 versions: A, B and C. These are built on top of the basic macro esk105. Each of these 3 macro's does something slightly different:

- A does not store any output pickles,
- B stores all output pickles,
- C starts at the 3rd chain of the macro.

Using these examples one can see how the way macro's are run can be controlled and what it saves to disk.

```
$ eskapade_run $TUTDIR/esk105_A_dont_store_results.py
$ eskapade_run $TUTDIR/esk105_B_store_each_chain.py
$ eskapade_run $TUTDIR/esk105_C_begin_at_chain3.py
```

Example esk106: Command line arguments

Macro 106 shows us how command line arguments can be used to control the chains in a macro. By adding the arguments from the message inside of the macro we can see that the chains are not run.

```
$ eskapade_run $TUTDIR/esk106_cmdline_options.py
```

Example esk107: Chain loop

Example 107 adds a chain to the macro and using a repeater Link it repeats the chain 10 times in a row.

```
$ eskapade_run $TUTDIR/esk107_chain_looper.py
```

Example esk108: Event loop

Example 108 processes a textual data set, to loop through every word and do a Map and Reduce operation on the data set. Finally a line printer prints out the result.

```
$ source $TUTDIR/esk108_eventlooper.sh
```

Example esk109: Debugging tips

This macro illustrates basic debugging features of Eskapade. The macro shows how to start a python session while running through the chains, and also how to break out of a chain.

```
$ eskapade_run $TUTDIR/esk109_debugging_tips.py
```

Example esk110: Code profiling

This macro demonstrates how to run Eskapade with code profiling turned on.

```
$ eskapade_run $TUTDIR/esk110_code_profiling.py
```

Example esk201: Read data

Macro 201 illustrates how to open files as pandas datasets. It reads a file into the DataStore. The first chain reads one csv into the DataStore, the second chain reads multiple files (actually the same file multiple times) into the DataStore. (Looping over data is shown in example esk209.)

```
$ eskapade_run $TUTDIR/esk201_readdata.py
```

Example esk202: Write data

Macro 202 illustrate writing pandas dataframes to file. It reads a DataFrame into the data store and then writes the DataFrame to csv format on the disk.

```
$ eskapade_run $TUTDIR/esk202_writedata.py
```

Example esk203: apply func to pandas df

Illustrates the link that calls basic apply() to columns of a pandas dataframes. See for more information pandas documentation:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.apply.html

```
$ eskapade_run $TUTDIR/esk203_apply_func_to_pandas_df.py
```

Example esk204: apply query to pandas df

Illustrates the link that applies basic queries to pandas dataframe. See for more information pandas documentation:

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.query.html

```
$ eskapade_run $TUTDIR/esk204_apply_query_to_pandas_df.py
```

Example esk205: concatenate pandas dfs

Illustrates the link that calls basic concat() of pandas dataframes. See for more information pandas documentation:

http://pandas.pydata.org/pandas-docs/stable/merging.html

```
$ eskapade_run $TUTDIR/esk205_concatenate_pandas_dfs.py
```

Example esk206: merge pandas dfs

Illustrate link that calls basic merge() of pandas dataframes. For more information see pandas documentation:

http://pandas.pydata.org/pandas-docs/stable/merging.html

```
$ eskapade_run $TUTDIR/esk206_merge_pandas_dfs.py
```

Example esk207: record vectorizer

This macro performs the vectorization of an input column of an input dataframe. E.g. a columnn x with values 1, 2 is tranformed into columns x_1 and x_2, with values True or False assigned per record.

```
$ eskapade_run $TUTDIR/esk207_record_vectorizer.py
```

Example esk208: record factorizer

This macro performs the factorization of an input column of an input dataframe. E.g. a columnn x with values 'apple', 'tree', 'pear', 'apple', 'pear' is tranformed into columns x with values 0, 1, 2, 0, 2.

```
$ eskapade_run $TUTDIR/esk208_record_factorizer.py
```

Example esk209: read big data itr

Macro to that illustrates how to loop over multiple (possibly large!) datasets in chunks.

```
$ eskapade_run $TUTDIR/esk209_read_big_data_itr.py
```

Example esk301: dfsummary plotter

Macro shows how to plot the content of a dataframe in a nice summary pdf file. (Example similar to `$TUTDIR/tutorial_1.py`.)

```
$ eskapade_run $TUTDIR/esk301_dfsummary_plotter.py
```

Example esk302: histogram_filler_plotter

Macro that illustrates how to loop over multiple (possibly large!) datasets in chunks, in each loop fill a (common) histogram, and plot the final histogram.

```
$ eskapade_run $TUTDIR/esk302_histogram_filler_plotter.py
```

Example esk303: histogrammar filler plotter

Macro that illustrates how to loop over multiple (possibly large!) datasets in chunks, in each loop fill a histogrammar histograms, and plot the final histograms.

```
$ eskapade_run $TUTDIR/esk302_histogram_filler_plotter.py
```

Example esk304: df boxplot

Macro shows how to boxplot the content of a dataframe in a nice summary pdf file.

```
$ eskapade_run $TUTDIR/esk304_df_boxplot.py
```

Example esk305: correlation summary

Macro to demonstrate generating nice correlation heatmaps using various types of correlation coefficients.

```
$ eskapade_run $TUTDIR/esk305_correlation_summary.py
```

Example esk306: concatenate reports

This macro illustrates how to concatenate the reports of several visualization links into one big report.

```
$ eskapade_run $TUTDIR/esk306_concatenate_reports.py
```

Example esk501: fix pandas dataframe

Macro illustrates how to call FixPandasDataFrame link that gives columns consistent names and datatypes. Default settings perform the following clean-up steps on an input dataframe:

- Fix all column names. Eg. remove punctuation and strange characters, and convert spaces to underscores.
- Check for various possible nans in the dataset, then make all nans consistent by turning them into numpy.nan (= float)
- Per column, assess dynamically the most consistent datatype (ignoring all nans in that column). Eg. bool, int, float, datetime64, string.
- Per column, make the data types of all rows consistent, by using the identified (or imposed) data type (by default ignoring all nans)

```
$ eskapade_run $TUTDIR/esk501_fix_pandas_dataframe.py
```

6.3.7 Tips on coding

This section contains a general description on how to use Eskapade in combination with other tools, in particular for the purpose of developing code.

Eskapade in PyCharm

PyCharm is a very handy IDE for debugging Python source code. It can be used to run Eskapade stand-alone (i.e. like from the command line) and with an API.

Stand-alone

- Install PyCharm on your machine.
- Open project and point to the Eskapade source code
- **Configuration, in 'Preferences', check the following desired values:**
  - **Under 'Project: eskapade' / 'Project Interpreter':**
    - The correct Python version (Python 3)
  - **Under 'Build, Execution & Deployment' / 'Console' / 'Python Console':**
    - The correct Python version (Python 3)
- Install Eskapade in editable mode
- **Run/Debug Configuration:**
  - Under 'Python' add new configuration
  - Script: path to the console script `eskapade_run` (located in the same directory as the interpreter specified above in 'Project Interpreter')

  - Script parameters: path to a macro to be debugged, e.g. `$ESKAPADE/python/eskapade/tutorials/tutorial_1.py`, and `eskapade_run` command line arguments, e.g. `--begin-with=Summary`
  - Python interpreter: check if it is the correct Python version (Python 3)

You should now be able to press the ‘play button’ to run Eskapade with the specified parameters.

Writing a new Link using Jupyter and notebooks

Running the framework works best from the command line (in our experience), but running experiments and trying new ideas is better left to an interactive environment like Jupyter. How can we reconcile the difference in these work flows? How can we use them together to get the most out of it?

Well, when using the data and config import functionality of Eskapade together with Jupyter we can interactively work on our objects and when we are satisfied with the results integration into links is straight-forward. The steps to undertake this are *in general* the following:

1. Import the DataStore and/or ConfigObject. Once you have imported the ConfigObject, run it to generate the output you want to use.
2. Grab the data you want from the DataStore using `ds = DataStore.import_from_file` and `data = ds[key]`.
3. Now you can apply the operation you want to do on the data, experiment on it and work towards the end result you want to have.
4. Create a new link in the appropriate link folder using the eskapade_generate_link command.
5. Copy the operations (code) you want to do to the link.
6. Add assertions and checks to make sure the Link is safe to run.
7. Add the Link to your macro and run it!

These steps are very general and we will now go into steps 5, 6 and 7. Steps 1, 2, 3 and 4 have already been covered by various parts of the documentation.

So assuming you wrote some new functionality that you want to add to a Link called YourLink and you have created a new Link from the template we are going to describe how you can get your code into the Link and test it.

Developing Links in notebooks

This subsection starts with a short summary of the workflow for developing Links:

1. Make your code in a notebook
2. Make a new Link
3. Port the code into the Link
4. Import the Link into your notebook
5. Test if the Link has the desired effect.
6. Finish the Link code
7. Write a unit test (optional but advised if you want to contribute)

We continue with a longer description of the steps above.

When adding the new code to a new link the following conventions are used:

In the `__init__` you specify the key word arguments of the Link and their default values, if you want to get an object from the DataStore or you want to write an object back into it, use the name `read_key` and `store_key`. Other keywords are free to use as you see fit.

In the `initialize` function in the Link you define and initialize functions that you want to call when executing the code on your objects. If you want to import something, you can do this at the root of the Link, as per PEP8.

In the `execute` function you put the actual code in this format:

```
settings = process_manager.service(ConfigObject)
ds = process_manager.service(DataStore)

## --- your code follows here
```

Now you can call the objects that contain all the settings and data of the macro in your Link, and in the code below you can add your analysis code that calls from the objects and writes back in case that this is necessary. Another possibility is writing a file to the disk, for example writing out a plot you made.

If you quickly want to test the Link without running the entire Eskapade framework, you can import it into your notebook sessions:

```
import eskapade.analysis.links.yourlink
from yourlink import YourLink
l = YourLink()
l.execute()
```

should run your link. You can also call the other functions. However, `execute()` is supposed to contain the bulk of your operations, so running that should give you your result. Now you can change the code in your link if it is not how you want it to run. The notebook kernel however keeps everything in memory, so you either have to restart the kernel, or use

```
import importlib
importlib.reload(eskapade.analysis.links.yourlink)
from yourlink import YourLink
l = YourLink()
l.execute()
```

to reload the link you changed. This is equivalent to the Python2 function `reload(eskapade).`

Combined with the importing of the other objects it becomes clear that you can run every piece of the framework from a notebook. However working like this is only recommended for development purposes, running an entire analysis should be done from the command line.

Finally after finishing all the steps you use the function `finalize()` to clean up all objects you do not want to save.

After testing whether the Link gives the desired result you have to add the proper assertions and other types of checks into your Link code to make sure that it does not have use-cases that are improperly defined. It is advised that you also write a unit test for the Link, but unless you want it merged into the master, it will not be enforced.

Now you can run Eskapade with your macro from your command line, using the new functionality that you first created in a notebook and then ported into a stand-alone Link.

6.4 Command Line Arguments

6.4.1 Overview

We start this section with a short overview of a few often used arguments of the Eskapade command `eskapade_run`. The only required argument is a configuration file, which can be a Python script (Eskapade macro) or a pickled Eskapade configuration object. This section gives an overview of the optional arguments of the run command.

At the end of running the Eskapade program, by default the DataStore and configuration object are pickled and written out to:

```
$ ls -l results/Tutorial_1/proc_service_data/v0/latest/
```

When you are working on a macro, once you are done tweaking it, you can also store the results of each chain in pickle files:

```
$ eskapade_run --store-all python/eskapade/tutorials/tutorial_1.py
```

Eskapade uses these pickle files to load the trained models and uses them to predict new samples real-time, but also to pick up running at a later stage in the chain setup.

For example, if running Eskapade takes a long time, you can run one chain as well:

```
$ eskapade_run --single-chain=Data python/eskapade/tutorials/tutorial_1.py
```

This command uses as input the stored pickle files from the previous chain. This might come in handy when, for example, data pre-processing of your training set takes a long time. In that case, you can run the pre-processing chain over night, store the results in a pickle file and start with the training chain the next day.

Start running Eskapade from a specified chain:

```
$ eskapade_run --begin-with=Summary python/eskapade/tutorials/tutorial_1.py
```

Stop running after a specified chain:

```
$ eskapade_run --end-with=Data python/eskapade/tutorials/tutorial_1.py
```

Below the most important command-line options are explained in detail.

6.4.2 Table of all arguments

The following table summarizes the available command-line options. Most of these options set variables in the Eskapade configuration object and can be overwritten by settings in the configuration macro.

Option	Short option	Argument	Description
–help	-h		show help message and exit
–analysis-name	-n	NAME	set name of analysis in run
–analysis-version	-v	VERSION	set version of analysis version in run
–batch-mode			run in batch mode (no X Windows)
–interactive	-i		start Python shell after run
–log-level	-L	LEVEL	set logging level
–log-format		FORMAT	set log-message format
–unpickle-config			interpret first CONFIG_FILE as path to pickled settings
–profile			run profiler for Python code
–conf-var	-c	KEY=VALUE	set configuration variable
–begin-with	-b	CHAIN_NAME	begin execution with chain CHAIN_NAME
–end-with	-e	CHAIN_NAME	end execution with chain CHAIN_NAME
–single-chain	-s	CHAIN_NAME	only execute chain CHAIN_NAME
–store-all			store run-process services after every chain
–store-one		CHAIN_NAME	store run-process services after chain CHAIN_NAME
–store-none			do not store run-process services
–results-dir		RESULTS_DIR	set directory path for results output
–data-dir		DATA_DIR	set directory path for data
–macros-dir		MACROS_DIR	set directory path for macros
–templates-dir		TEMPLATES_DIR	set directory path for template files
–spark-cfg-file		SPARK_CONFIG_FILE	set path of Spark configuration file
–seed		SEED	set seed for random-number generation

6.4.3 Description and examples

This section contains the most used options with a longer description of what it does and how it works combined with examples.

Set log level

The log level is controlled with the `--log-level` option. For example, to set the log level to “debug”, add:

```
--log-level=DEBUG
```

to the command line:

```
$ eskapade_run -L DEBUG python/eskapade/tutorials/tutorial_1.py
```

The available log levels are:

```
NOTSET,
DEBUG,
INFO,
WARNING,
ERROR,
FATAL
```

They correspond to the appropriate POSIX levels.

When writing your own Link, these levels can be accessed with the logger module:

```
self.logger.debug('Text to be printed when logging at DEBUG level')
```

All output is done in this manner, never with the python print function, since this yields us more control over the process.

Help

Help can be called by running the following:

```
$ eskapade_run --help
```

Interactive python mode

To keep the results in memory at end of session and access them in an interactive session, run Eskapade in interactive mode. This is controlled with `--interactive`:

```
$ eskapade_run -i python/eskapade/tutorials/tutorial_1.py
```

At the end of the session a `Python` console is started from which e.g. the data store can be accessed.

Saving states

To write out the intermediate results from every chain, add the command line argument `--store-all`. This will write pickles in `results/NAME/proc_service_data/VERSION/`, containing the state of Eskapade at the end of the chain:

```
$ eskapade_run --store-all python/eskapade/tutorials/tutorial_1.py
```

To write out the state after a particular chain, use option `--store-one`:

```
$ eskapade_run --store-one=Data python/eskapade/tutorials/tutorial_1.py
```

To not store any pickle files, run with the option `--store-none`:

```
$ eskapade_run --store-none python/eskapade/tutorials/tutorial_1.py
```

Single Chain

To run a single chain, use the option `--single-chain`. This picks up the data stored by the previous chain in the macro. It is, therefore, necessary to have run the previous chain, otherwise the engine can not start:

```
$ eskapade_run -s Summary python/eskapade/tutorials/tutorial_1.py
```

Start from a Chain

To start from a chain use the command line argument `--begin-with`. This picks up the data stored by the previous chain in the macro.

```
$ eskapade_run -b Summary python/eskapade/tutorials/tutorial_1.py
```

Stop at a Chain

To end the running of the engine at a chain use, the command line argument `--end-with`:

```
$ eskapade_run -e Data python/eskapade/tutorials/tutorial_1.py
```

Changing analysis version

A version number is assigned to each analysis, which by default is 0. It can be upgraded by using the option `--analysis-version`. When working on an analysis, it is recommended to update this number regularly for bookkeeping purposes. The command line always has higher priority over the macro. If the macro is version 0 and the command line uses version 1, the command line will overrule the macro.

```
$ eskapade_run -v 1 python/eskapade/tutorials/tutorial_1.py
```

Notice that the output of this analysis is now stored in the directory:

```
$ ls -l results/Tutorial_1/data/v1/report/
```

Notice as well that, for bookkeeping purposes, a copy of the (evolving) configuration macro is always stored as well, under:

```
$ ls -l results/Tutorial_1/config/v1/tutorial_1.py
```

Running an old configuration (macro)

Settings for the Eskapade run are stored in a configuration object, which is accessed as a run-process service. This run-time service can be persisted as a file, which is normally done at the end of the run.

Persisted settings can be used in a following run by providing the file path of the `ConfigObject` pickle file as the configuration file argument. The option `--unpickle-config` is required to indicate that this file contains persisted settings:

```
$ eskapade_run --unpickle-config results/Tutorial_1/proc_service_data/v0/latest/
↪eskapade.core.process_services.ConfigObject.pkl
```

In this way, rolling back to a previous point is straight-forward.

For lookup purposes a copy of the configuration macro is always stored as well, under:

```
$ ls -l results/Tutorial_1/config/v0/tutorial_1.py
```

Profiling your code

Your can profile the execution of your analysis functions with the option `--profile`:

```
$ eskapade_run --profile=cumulative python/eskapade/tutorials/tutorial_1.py
```

After running this prints out a long list of all functions called, including the time it took to run each of of them, where the functions are sorted based on cumulative time.

To get the the list of sorting options for the profiling, run:

```
$ eskapade_run --help
```

Combining arguments

Of course you can add multiple arguments to the command line, the result would be for example an interactive session in debug mode that writes out intermediate results from each chain:

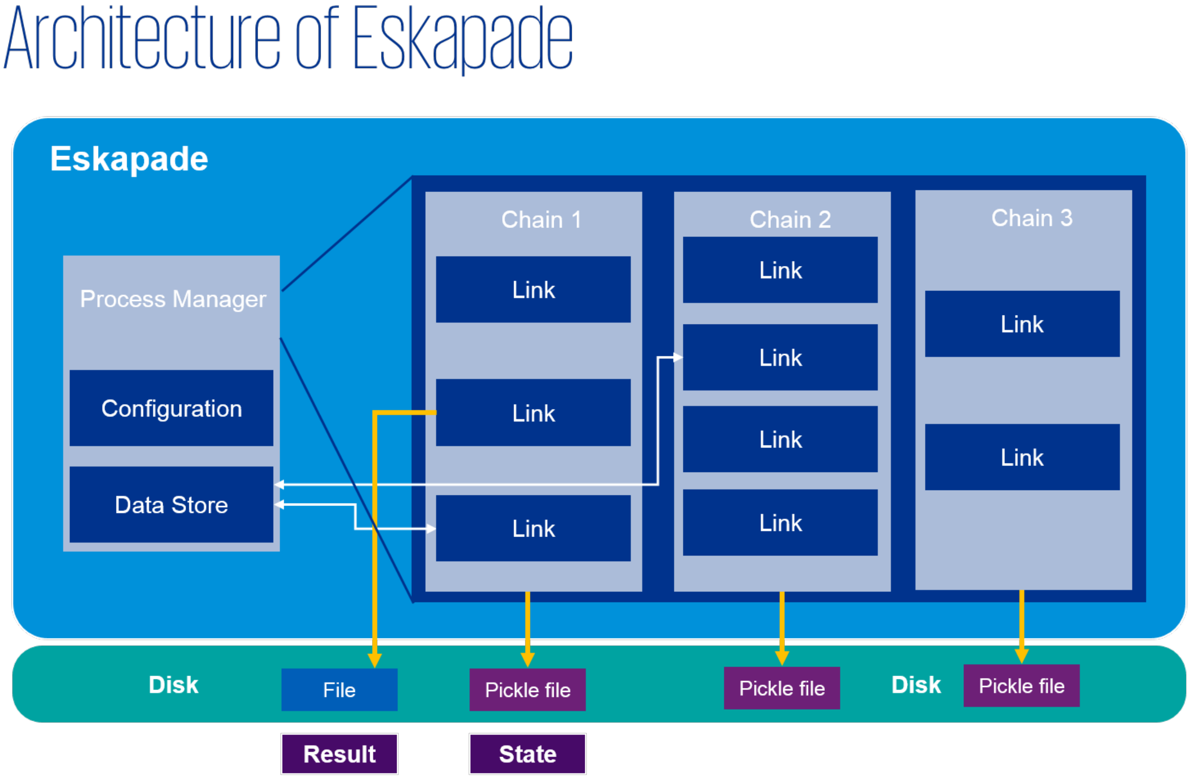
```
$ eskapade_run -i --store-all -L DEBUG -c do_chain0=False -c mydict="{'f': 'y=pi', 'pi
↪': 3.14}" python/eskapade/tutorials/esk106_cmdline_options.py
```

6.5 Package structure

Eskapade contains many tools, and to find and use them most efficiently it is necessary to understand how the repository is build up. This section discusses the structure of the code and how the framework handles subpackages.

6.5.1 Architecture

The architecture of Eskapade can be summarized in this picture:


Architecture of Eskapade
Eskapade
Process Manager
Configuration
Data Store
Chain 1
Link
Link
Link
Chain 2
Link
Link
Link
Link
Chain 3
Link
Link
Disk
File
Pickle file
Pickle file
Disk
Pickle file
Result
State

The example we just discussed generally shows how the framework works. The steps it takes are the following:

- eskapade_run runs the macro file,
- Macros (python file) contain Chains,
- Chains (python object) contains Links,
- Links (python class) contain analysis code.

The chains are run in the order of 'registering' them in the `ProcessManager`.

The `ProcessManager` is the ultimate object that executes all the code in your macro. It also keeps track of the configuration of Eskapade, and of the objects in the `data store` that are passable between links.

The components of the architecture of Eskapade are explained in further detail in the Tutorials section.

6.5.2 Structure

When using Eskapade it is important to understand where all components are located. The components can be for example links or utilities that you want to use.

The Eskapade framework is contained in the Python package `eskapade`, which lives in the `python` directory. Every specific subject has its subpackage in `eskapade`, containing the utilities it needs, the links that are defined for the subject.

The core of the framework is implemented in the `core` subpackage. This subpackage contains the low-level machinery for running analysis algorithms in chains of links. The `core_ops` subpackage contains basic links for operating this framework.

An example of a typical subpackage is `eskapade.analysis`, which contains basic analysis tools. Its structure is common to all Eskapade subpackages:

```
|-eskapade
   |-analysis
      |-links
```

The subpackage contains several modules, which contain classes and functions to be applied in links. The `eskapade.analysis.statistics` module, for example, contains code to generate an overview of the statistical properties of variables in given input data.

Eskapade links are located in the `links` directory. There is a separate module for each link, defining the link class instance. By convention, the names of the module and class are both the link name, the former in snake case and the latter in camel case. For example, the module `read_to_df` defines the link class `ReadToDf`.

The tests are contained separately in the Python package `eskapade_python` under `tests` directory. Ideally, there is a test module for each (link) module in the Eskapade package. Optionally, integration tests are implemented in `integration`. For the `eskapade.analysis` package, there is the module `test_tutorial_macros` with integration tests that run the tutorial macros corresponding to this subpackage:

```
|-eskapade_python
   |-analysis
      |-integration
         |-test_tutorial_macros.py
```

6.5.3 Subpackages

Eskapade contains the following list of subpackages:

- `core` is the package that contains the core framework of Eskapade.
- `core_ops` contains links pertaining to the core functionality of Eskapade.
- `analysis` contains pandas links and code.
- `visualization` contains visualization code and plotter links.
- `data_quality` contains links and code for fixing messy data.

6.5.4 Imports

Main elements of the Eskapade framework are imported directly from the `eskapade` package. For example, the run-configuration object and the run-process manager are part of the core subpackage, but are imported by

```
from eskapade import process_manager, ConfigObject
```

Links are imported directly from their subpackage:

```
from eskapade.analysis import ReadToDf
```

In a macro, you can now instantiate and configure the `ReadToDf` link and add it to a chain in the process manager.

6.5.5 Results

Results of a macro are written out by default in the `results` directory. The analysis run is persisted in the results directory by the `analysis_name` given in the macro. This directory has the following structure:

- `config`: the configuration macro
- `proc_service_data`: persisted states of run-process services
- `data`: analysis results, such as graphs or a trained model

The data for each of these elements are stored by the analysis version, e.g. `v0`, `v1`, `v2`, etc. For example, the report produced by the tutorial `esk301_dfsummary_plotter` is saved in the directory `results/esk301_dfsummary_plotter/data/v0/report`.

6.5.6 Debugging

When building new Links or other functionality you will want to debug at some point. There are multiple ways to do this, because there are multiple ways of running the framework. A few ways are:

- Running in the terminal. In this scenario you have to work in a virtual environment (or adjust your own until it has all dependencies) and debug using the terminal output.
- Running in a notebook. This way the code is run in a notebook and you can gather the output from the browser.
- Running in a docker. The code is run in the docker and the repository is mounted into the container. The docker (terminal) returns output.
- Running in a VM. In this case you run the code in the VM and mount the code into the VM. The output can be gathered in the VM and processed in the VM.

In the first three options you want to use an IDE or text-editor in a ‘normal’ environment to debug your code and in the last option you can use an editor in the VM or outside of it.

6.5.7 Troubleshooting

The least error prone ways are docker and VMs, because they automatically have the dependencies set.

6.6 Release notes

6.6.1 Version 0.8

In version 0.8 of Eskapade (August 2018) the modules `root-analysis` and `spark-analysis` have been split off into separate packages called Eskapade-ROOT and Eskapade-Spark .

So the (core) Eskapade package no longer depends on ROOT and Spark, just on plain python packages. This make it much easier for people to try out the core functionality of Eskapade.

- To install Eskapade-ROOT and Eskapade-Spark, do:

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

  or check out the code from out github repository:

  ```
  $ git clone git@github.com:KaveIO/Eskapade-ROOT.git eskapade-root
  $ pip install -e eskapade-root/
  $ git clone git@github.com:KaveIO/Eskapade-Spark.git eskapade-spark
  $ pip install -e eskapade-spark/
  ```

  where in this example the code is installed in edit mode (option -e).

  You can now use these in Python with:

  ```
  import eskapadespark
  import esroofit
  ```

6.6.2 Version 0.7

Version 0.7 of Eskapade (February 2018) contains several major updates:

- The Eskapade code has been made pip friendly. One can now simply do:

  ```
  $ pip install Eskapade
  ```

  or check out the code from out github repository:

  ```
  $ git clone git@github.com:KaveIO/Eskapade.git
  $ pip install -e Eskapade/
  ```

  where in this example the code is installed in edit mode (option -e).

  You can now use Eskapade in Python with:

  ```
  import eskapade
  ```

  This change has resulted in some restructuring of the python directories, making the overall structure more transparent: all python code, including the tutorials, now fall under the (single) `python/` directory. Additionally, thanks to the pip convention, our prior dependence on environment variables (`$ESKAPADE`) has now been fully stripped out of the code.

- There has been a cleanup of the core code, removing obsolete code and making it better maintainable. This has resulted in a (small) change in the api of the process manager, adding chains, and using the logger. All tutorials and example macro files have been updated accordingly. See the migration section for detailed tips on migrating existing Eskapade code to version 0.7.
- All eskapade commands now start with the prefix `eskapade_`. All tutorials have been updated accordingly. We have the commands:
  - `eskapade_bootstrap`, for creating a new Eskapade analysis project. See this new tutorial for all the details.
  - `eskapade_run`, for running the Eskapade macros.
  - `eskapade_trail`, for running the Eskapade unit and integration tests.
  - `eskapade_generate_link`, `eskapade_generate_macro`, `eskapade_generate_notebook`, for generating a new link, macro, or Jupyter notebook respectively.

6.6.3 Version 0.6

The primary feature of version 0.6 (August 2017) is the inclusion of Spark, but this version also includes several other new features and analyses.

We include multiple Spark links and 10 Spark examples on:

- The configuration of Spark, reading, writing and converting Spark dataframes, applying functions and queries to dataframes, filling histograms and (very useful!) applying arbitrary functions (e.g. pandas) to groupby calls.

In addition we hade added:

- A ROOT analysis for studying and quantifying between sets of (non-)categorical and observables. This is useful for finding outliers in arbitrary datasets (e.g. surveys), and we include a tutorial of how to do this.
- A ROOT analysis on predictive maintenance that decomposes a distribution of time difference between malfunctions by fitting this multiple Weibull distributions.
- New flexible features to create and chain analysis reports with several analysis and visualization links.

6.6.4 Version 0.5

Our 0.5 release (May 2017) contains multiple new features, in particular:

- Support for ROOT, including multiple examples on using data analysis, fitting and simulation examples using RooFit.
- Histogram conversion and filling support, using ROOT, numpy, Histogrammar and Eskapade-internal histograms.
- Automated data-quality fixes for buggy columns datasets, including data type fixing and NaN conversion.
- New visualization utilities, e.g. plotting multiple types of (non-linear) correlation matrices and dendograms.
- And most importantly, many new and interesting example macros illustrating the new features above!

6.6.5 Version 0.4

In our 0.4 release (Feb 2017) we are releasing the core code to run the framework. It is written in python 3. Anyone can use this to learn Eskapade, build data analyses with the link-chain methodology, and start experiencing its advantages.

The focus of the provided documentation is on constructing a data analysis setup in Eskapade. Machine learning interfaces will be included in an upcoming release.

6.7 Developing and Contributing

6.7.1 Preliminaries

6.7.2 Working on Eskapade

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

First clone Eskapade.

```
git clone git@github.com:KaveIO/Eskapade.git eskapade
```

then

```
pip install -e eskapade
```

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
```

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

```
cd eskapade
eskapade_trial .
```

or

```
cd eskapade
python setup.py test
```

That's it.

6.7.3 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.

6.7.4 Tips and Tricks

- Enable auto reload in ipython:

```
%load_ext autoreload
```

this will reload modules before executing any user code.

6.8 References

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

6.9 API

6.9.1 API Documentation

Eskapade

eskapade package

Subpackages

eskapade.analysis package

Subpackages

eskapade.analysis.links package

Submodules

eskapade.analysis.links.apply_func_to_df module

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

Class: ApplyFuncToDf

Created: 2016/11/08

Description: Algorithm to apply one or more functions to a (grouped) dataframe column or to an entire 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 `eskapade.analysis.links.apply_func_to_df.`**`ApplyFuncToDf`**`(`***kwargs*`)`

Bases: *`eskapade.core.element.Link`*

Apply functions to data-frame.

Applies one or more functions to a (grouped) dataframe column or an entire dataframe. In the latter case, this can be done row wise or column wise. The input dataframe will be overwritten.

`__init__``(`***kwargs*`)`

Initialize link instance.

Parameters

- **`read_key`** (*`str`*) – data-store input key
- **`store_key`** (*`str`*) – data-store output key
- **`apply_funcs`** (*`list`*) – functions to apply (list of dicts) - 'func': function to apply - 'colout' (string): output column - 'colin' (string, optional): input column - 'entire' (boolean, optional): apply to the entire dataframe? - 'args' (tuple, optional): args for 'func' - 'kwargs' (dict, optional): kwargs for 'func' - 'groupby' (list, optional): column names to group by - 'groupbyColout' (string) output column after the split-apply-combine combination
- **`add_columns`** (*`dict`*) – columns to add to output (name, column)

`add_apply_func`(*func*, *out_column*, *in_column=''*, **args*, ***kwargs*)
Add function to be applied to dataframe.

`execute`()
Execute the link.

`groupbyapply`(*df*, *groupby_columns*, *applyfunc*, **args*, ***kwargs*)
Apply groupby to dataframe.

`initialize`()
Initialize the link.

eskapade.analysis.links.apply_selection_to_df module

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

Class: ApplySelectionToDf

Created: 2016/11/08

Description: Algorithm to apply queries to input 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 `eskapade.analysis.links.apply_selection_to_df.`**`ApplySelectionToDf`**(***kwargs*)
Bases: *`eskapade.core.element.Link`*

Applies queries with sub-selections to a pandas dataframe.

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

Input dataframe is not overwritten, unless instructed to do so in kwargs.

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. If not set read_key is overwritten.
- **`query_set`** (*`list`*) – list of strings, query expressions to evaluate in the same order, see pandas documentation
- **`select_columns`** (*`list`*) – column names to select after querying

- **`continue_if_failure`** (*`bool`*) – if True continues with next query after failure (optional)
- **`kwargs`** – all other key word arguments are passed on to the pandas queries.

`execute``()`

Execute the link.

Applies queries or column selection to a pandas DataFrame. Input dataframe is not overwritten, unless told to do so in kwargs.

1. Apply queries, in order of provided query list.
2. Select columns (if provided).

`initialize``()`

Initialize the link.

Perform checks on provided attributes.

eskapade.analysis.links.basic_generator module

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

Class: BasicGenerator

Created: 2017/02/26

Description: Link to generate random data with basic distributions

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` `eskapade.analysis.links.basic_generator.`**`BasicGenerator`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Generate data with basic distributions.

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **`key`** (*`str`*) – key of output data in data store
- **`columns`** (*`list`*) – output column names
- **`size`** (*`int`*) – number of variable values
- **`gen_config`** (*`dict`*) – generator configuration for each variable
- **`gen_seed`** (*`int`*) – generator random seed

`execute``()`

Execute the link.

`initialize``()`

Initialize the link.

eskapade.analysis.links.df_concatenator module

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

Class: DataFrameColumnRenamer

Created: 2016/11/08

Description: Algorithm to concatenate multiple pandas datadrames

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 `eskapade.analysis.links.df_concatenator.`**`DfConcatenator`**`(`*`**kwargs`*`)`

Bases: `eskapade.core.element.Link`

Concatenates multiple pandas datadrames.

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **read_keys** (*list*) – keys of pandas dataframes in the data store
- **ignore_missing_input** (*bool*) – Skip missing input datasets. If all missing, store empty dataset. Default is false.
- **kwargs** – all other key word arguments are passed on to pandas concat function.

`execute``()`

Execute the link.

Perform concatenation of multiple pandas datadrames.

`initialize``()`

Initialize the link.

eskapade.analysis.links.df_merger module

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

Class: DfMerger

Created: 2016/11/08

Description: Algorithm to Merges two pandas DataFrames

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 `eskapade.analysis.links.df_merger.`**`DfMerger`**`(`*`**kwargs`*`)`

Bases: `eskapade.core.element.Link`

Merges two pandas dataframes.

`__init__`(***kwargs*)

Initialize link instance.

Store the configuration of the link.

Parameters

- **name** (*str*) – name of link
- **input_collection1** (*str*) – datastore key of the first pandas.DataFrame to merge
- **input_collection2** (*str*) – datastore key of the second pandas.DataFrame to merge
- **output_collection** (*str*) – datastore key of the merged output pandas.DataFrame
- **how** (*str*) – merge modus. See pandas documentation.
- **on** (*list*) – column names. See pandas documentation.
- **columns1** (*list*) – column names of the first pandas.DataFrame. Only these columns are included in the merge. If not set, use all columns.
- **columns2** (*list*) – column names of the second pandas.DataFrame. Only these columns are included in the merge. If not set, use all columns.
- **remove_duplicate_cols2** (*bool*) – if True duplicate columns will be taken out before the merge (default=True)
- **kwargs** – all other key word arguments are passed on to the pandas merge function.

execute()

Perform merging of input dataframes.

initialize()

Perform basic checks on provided attributes.

eskapade.analysis.links.histogrammar_filler module

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

Class: HistogrammarFiller

Created: 2017/03/21

Description: Algorithm to fill histogrammar sparse-bin histograms. 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

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

class `eskapade.analysis.links.histogrammar_filler.`**HistogrammarFiller**(***kwargs*)

Bases: *eskapade.analysis.histogram_filling.HistogramFillerBase*

Fill histogrammar sparse-bin histograms.

Algorithm to fill histogrammar style sparse-bin and category histograms.

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/esk303_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')]}
```

`construct_empty_hist`(*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 **`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)

eskapade.analysis.links.random_sample_splitter module

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

Class: RandomSampleSplitter

Created: 2016/11/08

Description: Algorithm to randomly assign records to a number of classes

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` `eskapade.analysis.links.random_sample_splitter.`**`RandomSampleSplitter`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Link that randomly assigns records of an input dataframe to a number of classes.

After assigning classes does one of the following:

- splits the input dataframe into sub dataframes according classes and stores the sub dataframes into the datastore;
- add a new column with assigned classes to the dataframe.

Records are assigned randomly.

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`read_key`** (*`str`*) – key of data to read from datastore
- **`store_key`** (*`list`*) – keys of datasets to store in datastore. Number of sub samples equals length of store_key list (optional instead of 'column' and 'nclasses').
- **`column`** (*`str`*) – name of new column that specifies the randomly assigned class. Default is randomclass (optional instead of 'store_key').
- **`nclasses`** (*`int`*) – number of random classes. Needs to be set (optional instead of 'store_key').
- **`fractions`** (*`list`*) – list of fractions (0<fraction<1) of records assigned to the sub samples. Can be one less than n classes. Sum can be less than 1. Needs to be set.
- **`nevents`** (*`list`*) – list of number of random records assigned to the sub samples Can be one less than n classes (optional instead of 'fractions').

`execute``()`

Execute the link.

`initialize``()`

Check and initialize attributes of the link.

eskapade.analysis.links.read_to_df module

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

Class: ReadToDf

Created: 2016/11/08

Description: Algorithm to write pandas dataframes picked up from the datastore.

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 `eskapade.analysis.links.read_to_df.`**`ReadToDf`**`(`***kwargs*`)`

Bases: `eskapade.core.element.Link`

Reads input file(s) to a pandas dataframe.

You give the link a path where your file is located and some kwargs that go into a pandas DataFrame. The kwargs are passed into the file reader.

`__init__``(`***kwargs*`)`

Initialize link instance.

Store the configuration of link ReadToDf.

Parameters

- **`name`** (`str`) – Name given to the link
- **`path`** (`str`) – path of your file to read into pandas DataFrame .
- **`key`** (`str`) – storage key for the DataStore.
- **`reader`** – pandas reader is determined automatically. But can be set by hand, e.g. csv, xlsx.
- **`itr_over_files`** (`bool`) – Iterate over individual files, default is false. If false, are files are collected in one dataframe. NB chunksize takes priority!
- **`chunksize`** (`int`) – Default is none. If positive integer then will always iterate. chunksize requires pd.read_csv or pd.read_table.
- **`kwargs`** – all other key word arguments are passed on to the pandas reader.

`execute``()`

Execute the link.

Reads the input file(s) and puts the dataframe in the datastore.

`initialize``()`

Initialize the link.

`is_finished``()` → bool

Try to assess if looper is done iterating over files.

Assess if looper is done or if a next dataset is still coming up.

`latest_data_length``()`

Return length of current dataset.

`set_chunk_size``(`*size*`)`

Set chunksize setting.

Parameters **`size`** – chunk size

sum_data_length()

Return sum length of all datasets processed sofar.

`eskapade.analysis.links.read_to_df.`**pandasReader**(*path*, *reader*, **args*, ***kwargs*)

Pick the correct pandas reader.

Based on provided reader setting, or based on file extension.

eskapade.analysis.links.record_factorizer module

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

Class: RecordFactorizer

Created: 2016/11/08

Description: Algorithm to perform the factorization of an input column of an input dataframe. E.g. a columnn x with values 'apple', 'tree', 'pear', 'apple', 'pear' is tranformed into columns x with values 0, 1, 2, 0, 2, etc.

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 `eskapade.analysis.links.record_factorizer.`**RecordFactorizer**(***kwargs*)

Bases: *`eskapade.core.element.Link`*

Factorize data-frame columns.

Perform factorization of input column of an input dataframe. E.g. a columnn x with values 'apple', 'tree', 'pear', 'apple', 'pear' is tranformed into columns x with values 0, 1, 2, 0, 2, etc. Resulting dataset stored as new dataset. Alternatively, map transformed columns back to orginal format.

__init__(***kwargs*)

Initialize link instance.

Store and do basic check on the attributes of link RecordFactorizer

Parameters

- **read_key** (*str*) – key to read dataframe from the data store. Dataframe of records that is to be transformed.
- **columns** (*list*) – list of columns that are to be factorized
- **inplace** (*bool*) – replace original columns. Default is False. Overwrites store_key to read_key.
- **convert_all_categories** (*bool*) – if true, convert all catergory observables. Default is false.
- **convert_all_booleans** (*bool*) – if true, convert all boolean observables. Default is false.
- **map_to_original** (*dict*) – dictiorary or key to dictionary to map back factorized columns to original. map_to_original is a dict of dicts, one dict for each column.
- **store_key** (*str*) – store key of output dataFrame. Default is read_key + '_fact'. (optional)
- **sk_map_to_original** (*str*) – store key of dictiorary to map factorized columns to original. Default is 'key' + '_' + store_key + '_to_original'. (optional)

- **sk_map_to_factorized** (*str*) – store key of dictiorary to map original to factorized columns. Default is 'key' + '_' + read_key + '_to_factorized'. (optional)

execute()

Execute the link.

Perform factorization input columns 'columns' of input dataframe. Resulting dataset stored as new dataset. Alternatively, map transformed columns back to orginal format.

initialize()

Initialize the link.

Initialize and (further) check the assigned attributes of the RecordFactorizer

eskapade.analysis.links.record_vectorizer module

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

Class: RecordVectorizer

Created: 2016/11/08

Description: Algorithm to perform the vectorization of an input column of an input 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 eskapade.analysis.links.record_vectorizer.**RecordVectorizer**(***kwargs*)

Bases: *eskapade.core.element.Link*

Vectorize data-frame columns.

Perform vectorization of input column of an input dataframe. E.g. a columnn x with values 1, 2 is tranformed into columns x_1 and x_2, with values True or False assigned per record.

__init__(***kwargs*)

Initialize link instance.

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

Parameters

- **read_key** (*str*) – key to read dataframe from the data store. Dataframe of records that is to be transformed.
- **columns** (*list*) – list of columns that are to be vectorized
- **store_key** (*str*) – store key of output dataFrame. Default is read_key + '_vectorized'. (optional)
- **column_compare_with** (*dict*) – dict of unique items per column with which column values are compared. If not given, this is derived automatically from the column. (optional)
- **astype** (*type*) – store answer of comparison of column with value as certain type. Default is bool. (optional)

execute()

Execute the link.

Perform vectorization input column 'column' of input dataframe. Resulting dataset stored as new dataset.

initialize()
Initialize the link.

Initialize and (further) check the assigned attributes of RecordVectorizer.

eskapade.analysis.links.record_vectorizer.**record_vectorizer**(*df*, *column_to_vectorize*, *column_compare_set*, *astype=<class 'bool'>*)

Vectorize data-frame column.

Takes the new record that is already transformed and vectorizes the given columns.

Parameters

- **df** – dataframe of the new record to vectorize
- **column_to_vectorize** (*str*) – string, column in the new record to vectorize.
- **column_compare_set** (*list*) – list of values to compare the column with.

Returns dataframe of the new records.

eskapade.analysis.links.value_counter module

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

Class: ValueCounter

Created: 2017/03/02

Description: Algorithm to do value_counts() on single columns of a pandas dataframe, or groupby().size() on multiple columns, both returned as dictionaries. It is possible to do cleaning of these dicts by rejecting certain keys or removing inconsistent data types. Numeric and timestamp columns are converted to bin indices before the binning is applied. Results are stored as 1D Histograms or as ValueCounts objects.

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 eskapade.analysis.links.value_counter.**ValueCounter**(***kwargs*)
Bases: *eskapade.analysis.histogram_filling.HistogramFillerBase*

Count values in Pandas data frame.

ValueCounter does value_counts() on single columns of a pandas dataframe, or groupby().size() on multiple columns. Results of both are returned as same-style dictionaries.

Numeric and timestamp columns are converted to bin indices before the binning is applied. The binning can be provided as input.

It is possible to do cleaning of these dicts by rejecting certain keys or removing inconsistent data types. Results are stored as 1D Histograms or as ValueCounts objects.

Example is available in: tutorials/esk302_histogram_filling_plotting.py

__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_counts** (*str*) – key of output data to store ValueCounts objects in data store
- **store_key_hists** (*str*) – key of output data to store histograms in data store
- **columns** (*list*) – columns 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]},
>>>              'date': {'bin_width': np.timedelta64(30, 'D'),
>>>                       'bin_offset': np.datetime64('2010-01-04')}}
```

Parameters

- **var_dtype** (*dict*) – dict of datatypes of the columns to study from dataframe. If not provided, try to determine datatypes directy from dataframe.
- **store_at_finalize** (*bool*) – Store histograms and/or ValueCount object in datastore at finalize(), not at execute(). Useful when looping over datasets. Default is False.
- **drop_inconsistent_key_types** (*bool*) – cleanup histograms and/or ValueCount objects by removing alls bins/keys with inconsistent datatypes. By default compare with data types in var_dtype dictionary.
- **dict** (*drop_keys*) – dictionary used for dropping specific keys from created value_counts dictionaries

Example drop_keys dictionary is:

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

drop_inconsistent_keys (*columns*, *obj*)

Drop inconsistent keys.

Drop inconsistent keys from a ValueCounts or Histogram object.

Parameters

- **columns** (*list*) – columns key to retrieve desired datatypes
- **obj** (*object*) – ValueCounts or Histogram object to drop inconsistent keys from

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)

finalize ()

Finalize ValueCounter.

initialize()

Initialize the link.

process_and_store()

Make, clean, and store ValueCount objects.

process_columns(*df*)

Process columns before histogram filling.

Specifically, convert timestamp columns to integers and numeric variables are converted to indices

Parameters **df** – input (pandas) data frame

Returns output (pandas) data frame with converted timestamp columns

Return type pandas DataFrame

eskapade.analysis.links.write_from_df module

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

Class: WriteFromDf

Created: 2016/11/08

Description: Algorithm to write a DataFrame from the DataStore to disk

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 eskapade.analysis.links.write_from_df.**WriteFromDf**(***kwargs*)

Bases: *eskapade.core.element.Link*

Write a DataFrame from the DataStore to disk.

__init__(***kwargs*)

Store the configuration of the link.

Parameters

- **name** (*str*) – Name given to the link
- **key** (*str*) – the DataStore key
- **path** (*str*) – path where to save the DataFrame
- **writer** – file extension that can be written by a pandas writer function from pd.DataFrame. For example: ‘csv’
- **dictionary** (*dict*) – keys (as in the arg above) and paths (as in the arg above) it will write out all the keys to the associated paths.
- **add_counter_to_name** (*bool*) – if true, add an index to the output file name. Useful when running in loops. Default is false.
- **kwargs** – all other key word arguments are passed on to the pandas writers.

execute()

Execute the link.

Pick up the dataframe and write to disk.

initialize()

Initialize the link.

eskapade.analysis.links.write_from_df.**pandas_writer**(*path*, *writer*)

Pick the correct pandas writer.

Based on provided writer setting, or based on file extension.

Module contents

class eskapade.analysis.links.**ApplyFuncToDf**(***kwargs*)

Bases: `eskapade.core.element.Link`

Apply functions to data-frame.

Applies one or more functions to a (grouped) dataframe column or an entire dataframe. In the latter case, this can be done row wise or column wise. The input dataframe will be overwritten.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **read_key** (`str`) – data-store input key
- **store_key** (`str`) – data-store output key
- **apply_funcs** (`list`) – functions to apply (list of dicts) - 'func': function to apply - 'colout' (string): output column - 'colin' (string, optional): input column - 'entire' (boolean, optional): apply to the entire dataframe? - 'args' (tuple, optional): args for 'func' - 'kwargs' (dict, optional): kwargs for 'func' - 'groupby' (list, optional): column names to group by - 'groupbyColout' (string) output column after the split-apply-combine combination
- **add_columns** (`dict`) – columns to add to output (name, column)

add_apply_func(*func*, *out_column*, *in_column=''*, **args*, ***kwargs*)

Add function to be applied to dataframe.

execute()

Execute the link.

groupbyapply(*df*, *groupby_columns*, *applyfunc*, **args*, ***kwargs*)

Apply groupby to dataframe.

initialize()

Initialize the link.

class eskapade.analysis.links.**ApplySelectionToDf**(***kwargs*)

Bases: `eskapade.core.element.Link`

Applies queries with sub-selections to a pandas dataframe.

__init__(***kwargs*)

Initialize link instance.

Input dataframe is not overwritten, unless instructed to do so in kwargs.

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. If not set read_key is overwritten.
- **query_set** (*list*) – list of strings, query expressions to evaluate in the same order, see pandas documentation
- **select_columns** (*list*) – column names to select after querying
- **continue_if_failure** (*bool*) – if True continues with next query after failure (optional)
- **kwargs** – all other key word arguments are passed on to the pandas queries.

execute()

Execute the link.

Applies queries or column selection to a pandas DataFrame. Input dataframe is not overwritten, unless told to do so in kwargs.

1. Apply queries, in order of provided query list.
2. Select columns (if provided).

initialize()

Initialize the link.

Perform checks on provided attributes.

class eskapade.analysis.links.**BasicGenerator**(***kwargs*)

Bases: `eskapade.core.element.Link`

Generate data with basic distributions.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **key** (*str*) – key of output data in data store
- **columns** (*list*) – output column names
- **size** (*int*) – number of variable values
- **gen_config** (*dict*) – generator configuration for each variable
- **gen_seed** (*int*) – generator random seed

execute()

Execute the link.

initialize()

Initialize the link.

class eskapade.analysis.links.**DfConcatenator**(***kwargs*)

Bases: `eskapade.core.element.Link`

Concatenates multiple pandas datadrames.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **store_key** (*str*) – key of data to store in data store
- **read_keys** (*list*) – keys of pandas dataframes in the data store

- **ignore_missing_input** (*bool*) – Skip missing input datasets. If all missing, store empty dataset. Default is false.
- **kwargs** – all other key word arguments are passed on to pandas concat function.

execute()

Execute the link.

Perform concatenation of multiple pandas datadrames.

initialize()

Initialize the link.

class eskapade.analysis.links.**DfMerger**(***kwargs*)

Bases: `eskapade.core.element.Link`

Merges two pandas dataframes.

__init__(***kwargs*)

Initialize link instance.

Store the configuration of the link.

Parameters

- **name** (*str*) – name of link
- **input_collection1** (*str*) – datastore key of the first pandas.DataFrame to merge
- **input_collection2** (*str*) – datastore key of the second pandas.DataFrame to merge
- **output_collection** (*str*) – datastore key of the merged output pandas.DataFrame
- **how** (*str*) – merge modus. See pandas documentation.
- **on** (*list*) – column names. See pandas documentation.
- **columns1** (*list*) – column names of the first pandas.DataFrame. Only these columns are included in the merge. If not set, use all columns.
- **columns2** (*list*) – column names of the second pandas.DataFrame. Only these columns are included in the merge. If not set, use all columns.
- **remove_duplicate_cols2** (*bool*) – if True duplicate columns will be taken out before the merge (default=True)
- **kwargs** – all other key word arguments are passed on to the pandas merge function.

execute()

Perform merging of input dataframes.

initialize()

Perform basic checks on provided attributes.

class eskapade.analysis.links.**HistogrammarFiller**(***kwargs*)

Bases: `eskapade.analysis.histogram_filling.HistogramFillerBase`

Fill histogrammar sparse-bin histograms.

Algorithm to fill histogrammar style sparse-bin and category histograms.

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/esk303_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')]}
```

`construct_empty_hist`(*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 **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)

class `eskapade.analysis.links.`**RandomSampleSplitter**(***kwargs*)

Bases: `eskapade.core.element.Link`

Link that randomly assigns records of an input dataframe to a number of classes.

After assigning classes does one of the following:

- splits the input dataframe into sub dataframes according classes and stores the sub dataframes into the datastore;
- add a new column with assigned classes to the dataframe.

Records are assigned randomly.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of data to read from datastore
- **store_key** (*list*) – keys of datasets to store in datastore. Number of sub samples equals length of store_key list (optional instead of 'column' and 'nclasses').
- **column** (*str*) – name of new column that specifies the randomly assigned class. Default is randomclass (optional instead of 'store_key').
- **nclasses** (*int*) – number of random classes. Needs to be set (optional instead of 'store_key').
- **fractions** (*list*) – list of fractions (0<fraction<1) of records assigned to the sub samples. Can be one less than n classes. Sum can be less than 1. Needs to be set.
- **nevents** (*list*) – list of number of random records assigned to the sub samples Can be one less than n classes (optional instead of 'fractions').

execute()

Execute the link.

initialize()

Check and initialize attributes of the link.

class `eskapade.analysis.links.`**ReadToDf**(***kwargs*)

Bases: `eskapade.core.element.Link`

Reads input file(s) to a pandas dataframe.

You give the link a path where your file is located and some kwargs that go into a pandas DataFrame. The kwargs are passed into the file reader.

__init__(***kwargs*)

Initialize link instance.

Store the configuration of link ReadToDf.

Parameters

- **name** (*str*) – Name given to the link
- **path** (*str*) – path of your file to read into pandas DataFrame .

- **key** (*str*) – storage key for the DataStore.
- **reader** – pandas reader is determined automatically. But can be set by hand, e.g. csv, xlsx.
- **itr_over_files** (*bool*) – Iterate over individual files, default is false. If false, are files are collected in one dataframe. NB chunksize takes priority!
- **chunksize** (*int*) – Default is none. If positive integer then will always iterate. chunksize requires pd.read_csv or pd.read_table.
- **kwargs** – all other key word arguments are passed on to the pandas reader.

execute()

Execute the link.

Reads the input file(s) and puts the dataframe in the datastore.

initialize()

Initialize the link.

is_finished() → bool

Try to assess if looper is done iterating over files.

Assess if looper is done or if a next dataset is still coming up.

latest_data_length()

Return length of current dataset.

set_chunk_size(*size*)

Set chunksize setting.

Parameters **size** – chunk size

sum_data_length()

Return sum length of all datasets processed sofar.

class eskapade.analysis.links.**RecordFactorizer**(***kwargs*)

Bases: *eskapade.core.element.Link*

Factorize data-frame columns.

Perform factorization of input column of an input dataframe. E.g. a columnn x with values ‘apple’, ‘tree’, ‘pear’, ‘apple’, ‘pear’ is tranformed into columns x with values 0, 1, 2, 0, 2, etc. Resulting dataset stored as new dataset. Alternatively, map transformed columns back to orginal format.

__init__(***kwargs*)

Initialize link instance.

Store and do basic check on the attributes of link RecordFactorizer

Parameters

- **read_key** (*str*) – key to read dataframe from the data store. Dataframe of records that is to be transformed.
- **columns** (*list*) – list of columns that are to be factorized
- **inplace** (*bool*) – replace original columns. Default is False. Overwrites store_key to read_key.
- **convert_all_categories** (*bool*) – if true, convert all catergory observables. Default is false.
- **convert_all_booleans** (*bool*) – if true, convert all boolean observables. Default is false.

- **map_to_original** (*dict*) – dictiorary or key to dictionary to map back factorized columns to original. map_to_original is a dict of dicts, one dict for each column.
- **store_key** (*str*) – store key of output dataFrame. Default is read_key + '_fact'. (optional)
- **sk_map_to_original** (*str*) – store key of dictiorary to map factorized columns to original. Default is 'key' + '_' + store_key + '_to_original'. (optional)
- **sk_map_to_factorized** (*str*) – store key of dictiorary to map original to factorized columns. Default is 'key' + '_' + read_key + '_to_factorized'. (optional)

execute()

Execute the link.

Perform factorization input columns 'columns' of input dataframe. Resulting dataset stored as new dataset. Alternatively, map transformed columns back to orginal format.

initialize()

Initialize the link.

Initialize and (further) check the assigned attributes of the RecordFactorizer

class eskapade.analysis.links.**RecordVectorizer**(***kwargs*)

Bases: `eskapade.core.element.Link`

Vectorize data-frame columns.

Perform vectorization of input column of an input dataframe. E.g. a columnn x with values 1, 2 is tranformed into columns x_1 and x_2, with values True or False assigned per record.

__init__(***kwargs*)

Initialize link instance.

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

Parameters

- **read_key** (*str*) – key to read dataframe from the data store. Dataframe of records that is to be transformed.
- **columns** (*list*) – list of columns that are to be vectorized
- **store_key** (*str*) – store key of output dataFrame. Default is read_key + '_vectorized'. (optional)
- **column_compare_with** (*dict*) – dict of unique items per column with which column values are compared. If not given, this is derived automatically from the column. (optional)
- **astype** (*type*) – store answer of comparison of column with value as certain type. Default is bool. (optional)

execute()

Execute the link.

Perform vectorization input column 'column' of input dataframe. Resulting dataset stored as new dataset.

initialize()

Initialize the link.

Initialize and (further) check the assigned attributes of RecordVectorizer.

class eskapade.analysis.links.**ValueCounter**(***kwargs*)

Bases: `eskapade.analysis.histogram_filling.HistogramFillerBase`

Count values in Pandas data frame.

ValueCounter does value_counts() on single columns of a pandas dataframe, or groupby().size() on multiple columns. Results of both are returned as same-style dictionaries.

Numeric and timestamp columns are converted to bin indices before the binning is applied. The binning can be provided as input.

It is possible to do cleaning of these dicts by rejecting certain keys or removing inconsistent data types. Results are stored as 1D Histograms or as ValueCounts objects.

Example is available in: tutorials/esk302_histogram_filling_plotting.py

`__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_counts`** (*`str`*) – key of output data to store ValueCounts objects in data store
- **`store_key_hists`** (*`str`*) – key of output data to store histograms in data store
- **`columns`** (*`list`*) – columns 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]},
>>>              'date': {'bin_width': np.timedelta64(30, 'D'),
>>>                       'bin_offset': np.datetime64('2010-01-04')}}
```

Parameters

- **`var_dtype`** (*`dict`*) – dict of datatypes of the columns to study from dataframe. If not provided, try to determine datatypes directy from dataframe.
- **`store_at_finalize`** (*`bool`*) – Store histograms and/or ValueCount object in datastore at finalize(), not at execute(). Useful when looping over datasets. Default is False.
- **`drop_inconsistent_key_types`** (*`bool`*) – cleanup histograms and/or ValueCount objects by removing alls bins/keys with inconsistent datatypes. By default compare with data types in var_dtype dictionary.
- **`dict`** (*`drop_keys`*) – dictionary used for dropping specific keys from created value_counts dictionaries

Example drop_keys dictionary is:

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

`drop_inconsistent_keys`(*columns*, *obj*)

Drop inconsistent keys.

Drop inconsistent keys from a ValueCounts or Histogram object.

Parameters

- **`columns`** (*`list`*) – columns key to retrieve desired datatypes
- **`obj`** (*`object`*) – ValueCounts or Histogram object to drop inconsistent keys from

`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)

`finalize`()
Finalize ValueCounter.

`initialize`()
Initialize the link.

`process_and_store`()
Make, clean, and store ValueCount objects.

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

Specifically, convert timestamp columns to integers and numeric variables are converted to indices

Parameters **`df`** – input (pandas) data frame

Returns output (pandas) data frame with converted timestamp columns

Return type pandas DataFrame

class `eskapade.analysis.links.`**`WriteFromDf`**(***kwargs*)
Bases: *`eskapade.core.element.Link`*

Write a DataFrame from the DataStore to disk.

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

Parameters

- **`name`** (*`str`*) – Name given to the link
- **`key`** (*`str`*) – the DataStore key
- **`path`** (*`str`*) – path where to save the DataFrame
- **`writer`** – file extension that can be written by a pandas writer function from pd.DataFrame. For example: ‘csv’
- **`dictionary`** (*`dict`*) – keys (as in the arg above) and paths (as in the arg above) it will write out all the keys to the associated paths.
- **`add_counter_to_name`** (*`bool`*) – if true, add an index to the output file name. Useful when running in loops. Default is false.
- **`kwargs`** – all other key word arguments are passed on to the pandas writers.

`execute`()
Execute the link.

Pick up the dataframe and write to disk.

initialize()

Initialize the link.

Submodules

eskapade.analysis.correlation module

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

Created: 2018/06/23

Description: Convert Pearson correlation value into a chi2 value of a contingency test matrix of a bivariate gaussion, and vice-versa. Calculation uses scipy's mvn library.

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.

`eskapade.analysis.correlation.`**chi2_from_rho**(*rho*, *n*, *subtract_from_chi2=0*, *corr0=None*, *sx=None*, *sy=None*, *nx=-1*, *ny=-1*)

Calculate chi2-value of bivariate gauss having correlation value rho

Calculate no-noise chi2 value of bivar gauss with correlation rho, with respect to bivariate gauss without any correlation.

Returns float chi2 value

`eskapade.analysis.correlation.`**rho_from_chi2**(*chi2*, *n*, *nx*, *ny*, *sx=None*, *sy=None*)

correlation coefficient of bivariate gaussian derived from chi2-value

Chi2-value gets converted into correlation coefficient of bivariate gauss with correlation value rho, assuming giving binning and number of records. Correlation coefficient value is between 0 and 1.

Bivariate gaussian's range is set to [-5,5] by construction.

Returns float correlation coefficient

eskapade.analysis.datetime module

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

Classes: TimePeriod, FreqTimePeriod

Created: 2017/03/14

Description: Time period and time period with frequency.

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

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

class `eskapade.analysis.datetime.`**FreqTimePeriod**(***kwargs*)

Bases: `eskapade.analysis.datetime.TimePeriod`

Time period with frequency.

__init__(***kwargs*)

Initialize TimePeriod instance.

`dt_string`(*period_index*)
Convert period index into date/time string (start of period).

Parameters **`period_index`** (`int`) – specified period index value.

`freq`
Return frequency.

`period_index`(*dt*)
Return number of periods until date/time “dt” since 1970-01-01.

Parameters **`dt`** – specified date/time parameter

class `eskapade.analysis.datetime.`**`TimePeriod`**(***kwargs*)
Bases: *`eskapade.core.mixin.ArgumentsMixin`*

Time period.

`__init__`(***kwargs*)
Initialize TimePeriod instance.

`logger`
A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

(continues on next page)

(continued from previous page)

```
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

classmethod parse_date_time(*dt*)
Try to parse specified date/time.

Parameters dt – specified date/time

classmethod parse_time_period(*period*)
Try to parse specified time period.

Parameters period – specified period

period_index(*dt*)
Get number of periods until date/time "dt".

Parameters dt – specified date/time

class eskapade.analysis.datetime.**UniformTsTimePeriod**(***kwargs*)
Bases: *eskapade.analysis.datetime.TimePeriod*

Time period with offset.

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

offset
Get offset parameter.

period
Get period parameter.

period_index(*dt*)
Get number of periods until date/time "dt" since "offset", given specified "period".

Parameters dt – specified date/time

eskapade.analysis.histogram module

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

Classes: ValueCounts, BinningUtil, Histogram

Created: 2017/03/14

Description: Generic 1D Histogram class.

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

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

class eskapade.analysis.histogram.**BinningUtil**(***kwargs*)
Bases: `object`

Helper for interpreting bin specifications.

BinningUtil is a helper class used for interpreting bin specification dictionaries. It is a base class for the Histogram class.

`__init__`(***kwargs*)

Initialize link instance.

A bin_specs dictionary needs to be provided as input. bins_specs is a dict containing 'bin_width' and 'bin_offset' keys. In case bins widths are not equal, bin_specs contains 'bin_edges' (array) instead of 'bin_width' and 'bin_offset'. 'bin_width' and 'bin_offset' can be numeric or numpy timestamps.

Alternatively, bin_edges can be provided as input to bin_specs.

Example bin_specs dictionaries are:

```
>>> bin_specs = {'bin_width': 1, 'bin_offset': 0}
>>> bin_spect = {'bin_edges': [0, 2, 3, 4, 5, 7, 8]}
>>> bin_specs = {'bin_width': np.timedelta64(30, 'D'),
                 'bin_offset': np.datetime64('2010-01-04')}
```

Parameters

- **`bin_specs`** (*`dict`*) – dictionary contains 'bin_width' and 'bin_offset' numbers or 'bin_edges' array. Default is None.
- **`bin_edges`** (*`list`*) – array with numpy histogram style bin_edges. Default is None.

`bin_specs`

Get bin_specs dictionary.

Returns bin_specs dictionary

Return type dict

`get_bin_center`(*bin_label*)

Return bin center for a given bin index.

Parameters **`bin_label`** – bin label for which to find the bin center

Returns bin center, can be float, int, timestamp

`get_bin_edges`()

Return bin edges.

Returns bin edges

Return type array

`get_bin_edges_range`()

Return bin range determined from bin edges.

Returns bin range

Return type tuple

`get_left_bin_edge`(*bin_label*)

Return left bin edge for a given bin index.

Parameters **`bin_label`** – bin label for which to find the left bin edge

Returns bin edge, can be float, int, timestamp

`get_right_bin_edge`(*bin_label*)

Return right bin edge for a given bin index.

Parameters **`bin_label`** – bin label for which to find the right bin edge.

Returns bin edge, can be float, int, timestamp

`truncated_bin_edges`(*variable_range=None*)

Bin edges corresponding to a given variable range.

Parameters **`variable_range`** (*`list`*) – variable range used for finding the right bin edges array. Optional.

Returns truncated bin edges

Return type array

`value_to_bin_label`(*var_value*, *greater_equal=False*)

Return bin index for given bin value.

Parameters

- **`var_value`** – variable value for which to find the bin index
- **`greater_equal`** (*`bool`*) – for float, int, timestamp, return index of bin for which value falls in range [lower edge, upper edge). If set to true, return index of bin for which value falls in range [lower edge, upper edge]. Default if false.

Returns bin index

Return type int

class `eskapade.analysis.histogram.`**`Histogram`**(*counts*, ***kwargs*)

Bases: *`eskapade.analysis.histogram.BinningUtil`*, *`eskapade.core.mixin.ArgumentsMixin`*

Generic 1D Histogram class.

Histogram holds bin labels (name of each bin), value_counts (values of the histogram) and a variable name. The bins can be categoric or numeric, where numeric includes timestamps. In case of numeric bins, bin_specs is set. bins_specs is a dict containing bin_width and bin_offset. In case bins widths are not equal, bin_specs contains bin_edges instead of bin_width and bin_offset.

`__init__`(*counts*, ***kwargs*)

Initialize Histogram instance.

A bin_specs dictionary can be provided as input. bins_specs is a dict containing 'bin_width' and 'bin_offset' keys. In case bins widths are not equal, bin_specs contains 'bin_edges' (array) instead of 'bin_width' and 'bin_offset'. 'bin_width' and 'bin_offset' can be numeric or numpy timestamps.

Histogram counts can be specified as a ValueCounts object, a dictionary or a tuple:

- tuple: Histogram((bin_values, bin_edges), variable=<your_variable_name>)
- dict: a dictionary as comes out of pandas.series.value_counts() or pandas.Dataframe.groupby.size() over one variable.
- ValueCounts: a ValueCounts object contains a value_counts dictionary.

Example bin_specs dictionaries are:

```
>>> bin_specs = { 'bin_width': 1, 'bin_offset': 0 }
>>> bin_spect = { 'bin_edges': [0,2,3,4,5,7,8] }
>>> bin_specs = { 'bin_width': np.timedelta64(30,'D'),
                  'bin_offset': np.datetime64('2010-01-04') }
```

Parameters

- **`counts`** – histogram counts

- **bin_specs** (*dict*) – dictionary contains 'bin_width' and 'bin_offset' numbers or 'bin_edges' array (default is None)
- **variable** (*str*) – name of the variable represented by the histogram
- **datatype** (*type*) – data type of the variable represented by the histogram (optional)

bin_centers()

Return bin centers.

Returns array of the bin centers

Return type array

bin_edges()

Return numpy style bin_edges array with uniform binning.

Returns array of all bin edges

Return type array

bin_entries()

Return number of bin entries.

Return the bin counts of the known bins in the value_counts object.

Returns array of the bin counts

Return type array

bin_labels()

Return bin labels.

Returns array of all bin labels

Return type array

classmethod combine_hists(*hists*, *labels=False*, *rel_bin_width_tol=1e-06*, ***kwargs*)

Combine a set of histograms.

Parameters

- **hists** (*array*) – array of Histograms to add up.
- **labels** (*label*) – histograms to add up have labels? (else are numeric) Default is False.
- **variable** (*str*) – name of variable described by the summed-up histogram
- **rel_bin_width_tol** (*float*) – relative tolerance between numeric bin edges.

Returns summed up histogram

Return type *Histogram*

copy(***kwargs*)

Return a copy of this histogram.

Parameters **variable** (*str*) – assign new variable name

datatype

Data type of the variable represented by the histogram.

Returns data type

Return type type

get_bin_count(*bin_label*)

Get bin count for specific bin label.

Parameters **`bin_label`** – a specific key to find corresponding bin.

Returns bin counter value

Return type int

`get_bin_labels``()`
Return all bin labels.

Returns array of all bin labels

Return type array

`get_bin_range``()`
Return the bin range.

Returns tuple of the bin range found

Return type tuple

`get_bin_vals``(`*variable_range=None*, *combine_values=True*`)`
Get bin labels/edges and corresponding bin counts.

Bin values corresponding to a given variable range.

Parameters

- **`variable_range`** (*`list`*) – variable range used for finding the right bins to get values from. Optional.
- **`combine_values`** (*`bool`*) – if bin_specs is not set, combine existing bin labels with variable range.

Returns two arrays of bin values and bin edges

Return type array

`get_hist_val``(`*var_value*`)`
Get bin count for bin by value of histogram variable.

Parameters **`var_value`** – a specific value to find corresponding bin.

Returns bin counter value

Return type int

`get_nonone_bin_centers``()`
Return bin centers.

Returns array of the bin centers

Return type array

`get_nonone_bin_counts``()`
Return bin counts.

Returns array of the bin counts

Return type array

`get_nonone_bin_edges``()`
Return numpy style bin-edges array.

Returns array of the bin edges

Return type array

get_nonone_bin_range()

Return the bin range.

Returns tuple of the bin range found

Return type tuple

get_uniform_bin_edges()

Return numpy style bin-edges array with uniform binning.

Returns array of all bin edges

Return type array

logger

A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

`n_bins`

Number of bins in the ValueCounts object.

Returns number of bins

Return type int

`n_dim`

Number of histogram dimensions.

The number of histogram dimensions, which is equal to one by construction.

Returns number of dimensions

Return type int

`num_bins`

Number of bins.

Returns number of bins

Return type int

`remove_keys_of_inconsistent_type`(*prefered_key_type=None*)

Remove all keys that have inconsistent data type(s).

Parameters **`prefered_key_type`** (`tuple`) – the prefered key type to keep. Can be a tuple, list, or single type. E.g. str or (int,str,float). If None provided, the most common key type found is kept.

`simulate`(*size*, **args*)

Simulate data using self (Histogram instance) as PDF.

see https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.html

Parameters **`size`** (`int`) – number of data points to generate

Return numpy.array generated_data the generated data

Returns Histogram of the generated data

Return type *Histogram*

`surface`()

Calculate surface of the histogram.

Returns surface

`to_normalized`(***kwargs*)

Return a normalized copy of this histogram.

Parameters

- **`new_var_name`** (`str`) – assign new variable name
- **`variable_range`** (`list`) – variable range used for finding the right bins to get values from.
- **`combine_values`** (`bool`) – if bin_specs is not set, combine existing bin labels with variable range.

`variable`

Name of variable represented by the histogram.

Returns variable name

Return type string

class `eskapade.analysis.histogram.`**`ValueCounts`**(*key*, *subkey=None*, *counts=None*, *sel=None*)

Bases: `object`

A dictionary of value counts.

The dictionary of value counts comes out of pandas.series.value_counts() for one variable or pandas.Dataframe.groupby.size() performed over one or multiple variables.

`__init__`(*key*, *subkey=None*, *counts=None*, *sel=None*)

Initialize link instance.

Parameters

- **`key`** (*`list`*) – key is a tuple, list or string of (the) variable name(s), matching those and the structure of the keys in the value_counts dictionary.
- **`subkey`** (*`list`*) – subset of key. If provided, the value_counts dictionary will be projected from key onto the (subset of) subkey. E.g. use this to map a two dimensional value_counts dictionary onto one specified dimension. Default is None. Optional.
- **`counts`** (*`dict`*) – the value_counts dictionary.
- **`sel`** (*`dict`*) – Apply selections to value_counts dictionary. Default is {}. Optional.

`count`(*value_bin*)

Get bin count for specific bin-key value bin.

Parameters **`value_bin`** (*`tuple`*) – a specific key, and can be a list or tuple.

Returns specific bin counter value

Return type int

`counts`

Process and return value-counts dictionary.

Returns after processing, returns the value_counts dictionary

Return type dict

`create_sub_counts`(*subkey*, *sel=None*)

Project existing value counts onto a subset of keys.

E.g. map variables x,y onto single dimension x, so for each bin in x integrate over y.

Parameters

- **`subkey`** (*`tuple`*) – input sub-key, is a tuple, list, or string. This is the new key of variables for the returned ValueCounts object.
- **`sel`** (*`dict`*) – dictionary with selection. Optional.

Returns value_counts object where subkey has become the new key.

Return type *ValueCounts*

`get_values`(*val_keys=()*)

Get all key-values of a subset of keys.

E.g. give all x values in of the keys, when the value_counts object has keys (x, y).

Parameters **`val_keys`** (*`tuple`*) – a specific sub-key to get key values for.

Returns all key-values of a subset of keys.

Return type tuple

`key`

Process and return current value-counts key.

Returns the key

Return type tuple

`nononecounts`

Return value-counts dictionary without None keys.

Returns after processing, returns the value_counts dictionary without None keys

Return type dict

`num_bins`

Number of value-counts bins.

Returns number of bins

Return type int

`num_nonone_bins`

Number of not-none value-counts bins.

Returns number of not-none bins

Return type int

`process_counts`(*accept_equiv=True*)

Project value counts onto the existing subset of keys.

E.g. map variables x,y onto single dimension x, so for each bin in x integrate over y.

Parameters **`accept_equiv`** (*`bool`*) – accept equivalence of key and subkey if if subkey is in different order than key. Default is true.

Returns successful projection or not

Return type bool

`remove_keys_of_inconsistent_type`(*prefered_key_type=None*)

Remove keys with inconsistent data type(s).

Parameters **`prefered_key_type`** (*`tuple`*) – the prefered key type to keep. Can be a tuple, list, or single type. E.g. str or (int, str, float). If None provided, the most common key type found is kept.

`skey`

Current value-counts subkey.

Returns the subkey

Return type tuple

`sum_counts`

Sum of counts of all value-counts bins.

Returns the sum of counts of all bins

Return type float

`sum_nonone_counts`

Sum of not-none counts of all value-counts bins.

Returns the sum of not-none counts of all bins

Return type float

eskapade.analysis.histogram_filling module

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

Class: HistogramFillerBase

Created: 2017/03/21

Description: Algorithm to fill histogrammar sparse-bin histograms. 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

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

class `eskapade.analysis.histogram_filling.`**`HistogramFillerBase`**`(**kwargs)`

Bases: *`eskapade.core.element.Link`*

Base class link to fill histograms.

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.

`__init__``(**kwargs)`

Initialize link instance.

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

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.
- **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 pandas data frame.

Parameters **`df`** – input (pandas) data frame

`categorize_columns`(*df*)

Categorize columns of dataframe by data type.

Parameters **`df`** – input (pandas) data frame

`drop_requested_keys`(*name*, *counts*)

Drop requested keys from counts dictionary.

Parameters

- **`name`** (`string`) – key of drop_keys dict to get array of keys to be dropped
- **`counts`** (`dict`) – counts dictionary to drop specific keys from

Returns count dict without dropped keys

`execute`()

Execute the link.

Execute() four things:

- check presence and data type of requested columns
- timestamp variables are converted to nanosec (integers)
- do the actual value counting based on categories and created indices
- then convert to histograms and add to datastore

`fill_histogram`(*idf*, *c*)

Fill input histogram with column(s) of input dataframe.

Parameters

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

`finalize`()

Finalize the link.

Store Histograms here, if requested.

`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

initialize()
Initialize the link.

process_and_store()
Store (and possibly process) histogram objects.

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

Specifically, convert timestamp columns to integers

Parameters **df** – input (pandas) data frame

Returns output (pandas) data frame with converted timestamp columns

Return type pandas DataFrame

eskapade.analysis.histogram_filling.**only_bool**(*val*)
Pass input value or array only if it is a bool.

Parameters **val** – value to be evaluated

Returns evaluated value

Return type np.bool or np.ndarray

eskapade.analysis.histogram_filling.**only_float**(*val*)
Pass input val value or array only if it is a float.

Parameters **val** – value to be evaluated

Returns evaluated value

Return type np.float64 or np.ndarray

eskapade.analysis.histogram_filling.**only_int**(*val*)
Pass input val value or array only if it is an integer.

Parameters **val** – value to be evaluated

Returns evaluated value

Return type np.int64 or np.ndarray

eskapade.analysis.histogram_filling.**only_str**(*val*)
Pass input value or array only if it is a string.

Parameters **val** – value to be evaluated

Returns evaluated value

Return type str or np.ndarray

eskapade.analysis.histogram_filling.**to_ns**(*x*)
Convert input timestamps to nanoseconds (integers).

Parameters **x** – value to be converted

Returns converted value

Return type int

eskapade.analysis.histogram_filling.**to_str**(*val*)
Convert input to (array of) string(s).

Parameters **val** – value to be converted

Returns converted value

Return type str or np.ndarray

`eskapade.analysis.histogram_filling.`**`value_to_bin_center`**`(`*val*, ***kwargs*`)`

Convert value to bin center.

Convert a numeric or timestamp column to a common bin center value.

Parameters

- **`bin_width`** – bin_width value needed to convert column to a common bin center value
- **`bin_offset`** – bin_offset value needed to convert column to a common bin center value

`eskapade.analysis.histogram_filling.`**`value_to_bin_index`**`(`*val*, ***kwargs*`)`

Convert value to bin index.

Convert a numeric or timestamp column to an integer bin index.

Parameters

- **`bin_width`** – bin_width value needed to convert column to an integer bin index
- **`bin_offset`** – bin_offset value needed to convert column to an integer bin index

eskapade.analysis.statistics module

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

Classes: ArrayStats, GroupByStats

Created: 2017/03/21

Description: Summary of an array.

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` `eskapade.analysis.statistics.`**`ArrayStats`**`(`*data*, *col_name*, *weights=None*, *unit=”*, *label=”*`)`

Bases: `object`

Create summary of an array.

Class to calculate statistics (mean, standard deviation, percentiles, etc.) and create a histogram of values in an array. The statistics can be returned as values in a dictionary, a printable string, or as a LaTeX string.

`__init__``(`*data*, *col_name*, *weights=None*, *unit=”*, *label=”*`)`

Initialize for a single column in data frame.

Parameters

- **`data`** (*`(keys of) dict`*) – Input array
- **`col_name`** – column name
- **`weights`** (*`string (column of data)`*) – Input array (default None)
- **`unit`** – Unit of column
- **`label`** (*`str`*) – Label to describe column variable

Raises TypeError

`create_mpv_stat`()
Compute most probable value from histogram.

This function computes the most probable value based on the histogram from make_histogram(), and adds it to the statistics.

`create_stats`()
Compute statistical properties of column variable.

This function computes the statistical properties of values in the specified column. It is called by other functions that use the resulting figures to create a statistical overview.

`get_col_props`()
Get column properties.

Returns dict Column properties

`get_latex_table`(*get_stats=None*, *latex=True*)
Get LaTeX code string for table of stats values.

Parameters

- **`get_stats`** (*`list`*) – List of statistics that you want to filter on. (default None (all stats)) Available stats are: 'count', 'filled', 'distinct', 'mean', 'std', 'min', 'max', 'p05', 'p16', 'p50', 'p84', 'p95', 'p99'
- **`latex`** (*`bool`*) – LaTeX output or list output (default True)

Returns str LaTeX code snippet

`get_print_stats`(*to_output=False*)
Get statistics in printable form.

Parameters **`to_output`** (*`bool`*) – Print statistics to output stream?

Returns str Printable statistics string

`get_x_label`()
Get x label.

`logger`
A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
```

(continues on next page)

(continued from previous page)

```
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

make_histogram(*var_bins=30*, *var_range=None*, *bin_edges=None*, *create_mpv_stat=True*)
Create histogram of column values.

Parameters

- **var_bins** (*int*) – Number of histogram bins
- **var_range** (*tuple*) – Range of histogram variable
- **bin_edges** (*list*) – predefined bin edges to use for histogram. Overrules var_bins.

class eskapade.analysis.statistics.**GroupByStats**(*data*, *col_name*, *groupby=None*, *weights=None*, *unit=''*, *label=''*)
Bases: *eskapade.analysis.statistics.ArrayStats*

Create summary of an array in groups.

__init__(*data*, *col_name*, *groupby=None*, *weights=None*, *unit=''*, *label=''*)
Initialize for a single column in dataframe.

Parameters

- **data** (*(keys of) dict*) – Input array
- **col_name** – column name
- **weights** (*string (column of data)*) – Input array (default None)
- **unit** – Unit of column
- **label** (*str*) – Label to describe column variable
- **groupby** – column name

Raises TypeError

get_latex_table(*get_stats=None*)
Get LaTeX code string for group-by table of stats values.

Parameters **get_stats** (*list*) – same as ArrayStats.get_latex_table get_stats key word.

Returns str LaTeX code snippet

eskapade.analysis.statistics.**get_col_props**(*var_type*)
Get column properties.

Returns dict Column properties

eskapade.analysis.statistics.**weighted_quantile**(*data*, *weights=None*, *probability=0.5*)
Compute the weighted quantile of a 1D numpy array.

Weighted quantiles, inspired by: https://github.com/nudomarinero/wquantiles/blob/master/wquantiles.py written by Jose Sabater Here updated to return multiple quantiles in one go. Now also works when weight is None.

Parameters

- **data** (*ndarray*) – input array (one dimension).
- **weights** (*ndarray*) – array with the weights of the same size of *data*.
- **probability** (*ndarray*) – array of quantiles to compute. Each probablity must have a value between 0 and 1.

Returns list of the output value(s).

Module contents

eskapade.core package

Submodules

eskapade.core.definitions module

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

Created: 2017/02/27

Description: Definitions used in Eskapade runs: * logging levels * return-status codes * default configuration variables * user options

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 eskapade.core.definitions.**RandomSeeds**(***kwargs*)
Bases: `object`

Container for seeds of random generators.

Seeds are stored as key-value pairs and are accessed with getitem and setitem methods. A default seed can be accessed with the key “default”. The default seed is also returned if no seed is set for the specified key.

```
>>> import numpy as np
>>> seeds = RandomSeeds(default=999, foo=42, bar=13)
>>> seeds['NumPy'] = 100
>>> np.random.seed(seeds['NumPy'])
```

(continues on next page)

(continued from previous page)

```
>>> print(seeds['nosuchseed'])
999
```

__init__(***kwargs*)

Initialize an instance.

Values of the specified keyword arguments must be integers, which are set as seed values for the corresponding key.

class `eskapade.core.definitions.`**StatusCode**

Bases: `enum.IntEnum`

Return status code enumeration class.

A StatusCode should be returned by the initialize, execute, and finalize methods of links, chains, and the process manager.

The enumerations are:

- Undefined (-1): Default status.
- Success (0 == EX_OK / EXIT_SUCCESS): All OK, i.e. there were no errors.
- RepeatChain (1): Repeat execution of this chain.
- SkipChain (2): Skip this chain: initialize, execute, and finalize.
- BreakChain (3): Skip the further execution of this this, but do perform finalize.
- Recoverable (4): Not OK, but can continue, i.e. there was an error, but the application can recover from it.
- Failure (5): An error occurred and the application cannot recover from it. In this case the application should just quit.

BreakChain = 3

Failure = 5

Recoverable = 4

RepeatChain = 1

SkipChain = 2

Success = 0

Undefined = -1

`eskapade.core.definitions.`**set_begin_end_chain_opt**(*opt_key*, *settings*, *args*)

Set begin/end-chain variable from user option.

`eskapade.core.definitions.`**set_custom_user_vars**(*opt_key*, *settings*, *args*)

Set custom user configuration variables.

`eskapade.core.definitions.`**set_log_level_opt**(*opt_key*, *settings*, *args*)

Set configuration log level from user option.

`eskapade.core.definitions.`**set_opt_var**(*opt_key*, *settings*, *args*)

Set configuration variable from user options.

`eskapade.core.definitions.`**set_seeds**(*opt_key*, *settings*, *args*)

Set random seeds.

`eskapade.core.definitions.`**set_single_chain_opt**(*opt_key*, *settings*, *args*)

Set single-chain variable from user option.

eskapade.core.element module

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

Created: 2017/02/27

Description: Base classes for the building blocks of an Eskapade analysis run:

- Link:
- Chain:

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 `eskapade.core.element.`**`Chain`**(*name*, *process_manager=None*)

Bases: *`eskapade.core.meta.Processor`*, *`eskapade.core.meta.ProcessorSequence`*, *`eskapade.core.mixin.TimerMixin`*

Execution Chain.

A Chain object contains a collection of links with analysis code. The links in a chain are executed in the order in which the links have been added to the chain. Typically a chain contains all links related to one topic, for example 'validation of a model', or 'data preparation', or 'data quality checks'.

```
>>> from eskapade import process_manager
>>> from eskapade import Chain
>>> from eskapade import analysis
>>>
>>> # Create an IO chain. This is automatically registered with the process␣
↪manager.
>>> io_chain = Chain('Overview')
```

And Links are added to a chain as follows:

```
>>> # add a link to the chain
>>> io_chain.add(analysis.ReadToDf(path='foo.csv', key='foo'))
>>>
>>> # Run everything.
>>> process_manager.run()
```

`__init__`(*name*, *process_manager=None*)

Initialize chain.

`add`(*link: eskapade.core.element.Link*) → None

Add a link to the chain.

Parameters **`link`** (`Link`) – The link to add to the chain.

Raises

- **`TypeError`** – When the link is of an incompatible type.
- **`KeyError`** – When a Link of the same type and name already exists.

`clear`()

Clear the chain.

`discard`(*link: eskapade.core.element.Link*) → None

Remove a link from the chain.

Parameters **`link`** (`Link`) –

Raises **`KeyError`** – When the processor does not exist.

`execute`() → eskapade.core.definitions.StatusCode

Execute links in chain.

Returns Execution status code.

Return type *StatusCode*

`finalize`() → eskapade.core.definitions.StatusCode

Finalize links and chain.

Returns Finalization status code.

Rtype StatusCode

`get`(*link_name: str*) → eskapade.core.element.Link

Find the link with the given name.

Parameters **`link_name`** (*`str`*) – Find a link with the given name.

Returns The chain.

Return type *Chain*

Raises **`ValueError`** – When the given chain name cannot be found.

`initialize`() → eskapade.core.definitions.StatusCode

Initialize chain and links.

Returns Initialization status code.

Return type *StatusCode*

`n_links`

Return the number of links in the chain.

Returns The number of links in the chain.

Return type int

class `eskapade.core.element.`**`Link`**(*name=None*)

Bases: *`eskapade.core.meta.Processor`*, *`eskapade.core.mixin.ArgumentsMixin`*, *`eskapade.core.mixin.TimerMixin`*

Link base class.

A link defines the content of an algorithm. Any actual link is derived from this base class.

A link usually does three things: - takes data from the datastore - does something to it - writes data back

To take from the data store there is a simple function load() To write to the data store there is a simple function store()

Links are added to a chain as follows:

```
>>> from eskapade import process_manager
>>> from eskapade import analysis
>>>
>>> # Create a Chain instance. Note that the chain is automatically registered␣
↪with process manager.
>>> io_chain = Chain('IO')
>>>
>>> # Add a link to the chain
```

(continues on next page)

(continued from previous page)

```
>>> reader = analysis.ReadToDf(name='CsvReader', key='foo')
>>> reader.path = 'foo.csv'
>>> io_chain.add(reader)
>>>
>>> # Run everything.
>>> process_manager.run()
```

`__init__`(*name=None*)

Initialize link.

`execute`() → eskapade.core.definitions.StatusCode

Execute the Link.

This method may be overridden by the user.

Returns Status code.

Return type *StatusCode*

`finalize`() → eskapade.core.definitions.StatusCode

Finalize the Link.

This method may be overridden by the user.

Returns Status code.

Return type *StatusCode*

`initialize`() → eskapade.core.definitions.StatusCode

Initialize the Link.

This method may be overridden by the user.

Returns Status code.

Type StatusCode

`load`(*ds*, *read_key=None*)

Read all data from specified source.

Read_key can either be:

- one Link: return statuscode, [data_from_link,. . .]
- A list of locations: return statuscode, [data,. . .]
- A list of links with only one output location: return statuscode, [data,. . .]
- A list of links with multiple output locations: return statuscode, [data,[moredata]. . .]
- Any mixture of the above

Do something logical with a statuscode if this data does not exist link.if_input_missing = statuscode

Returns a tuple statuscode, [data in same order as read_key]

Return type (*StatusCode*,list)

`run`() → eskapade.core.definitions.StatusCode

Initialize, execute, and finalize the Link in one go.

This method is useful for testing purposes, e.g. when developing and testing functionality of a link standalone and interactively.

It is not used internally by Eskapade, where the functions are called individually by the chain, and all links are initialized together before their common execution, and all links in the chain are also finalized together, after their common execution.

Returns Status code.

Return type *StatusCode*

`store`(*ds*, *data*, *store_key=None*, *force=False*)

Store data back to datastore.

Do something logical with a statuscode if this data already exists link.if_output_exists = statuscode uses self.store_key. If self.store_key is a list of locations, I must sent a list of the same length here

`summary`()

Print a summary of the main settings of the link.

eskapade.core.exceptions module

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

Created: 2016/11/08

Description: Eskapade exceptions.

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.

exception `eskapade.core.exceptions.`**`Error`**

Bases: `Exception`

Base class for all Eskapade core exceptions.

exception `eskapade.core.exceptions.`**`UnknownSetting`**

Bases: *`eskapade.core.exceptions.Error`*

The user requested an unknown setting.

eskapade.core.execution module

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

Created: 2016/11/08

Description: Functions for running and resetting Eskapade machinery

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.

`eskapade.core.execution.`**`eskapade_run`**(*settings=None*)

Run Eskapade.

This function is called in the script eskapade_run when run from the cmd line. The working principle of Eskapade is to run chains of custom code chunks (so-called links).

Each chain should have a specific purpose, for example pre-processing incoming data, booking and/or training predictive algorithms, validating these predictive algorithms, evaluating the algorithms.

By using this principle, links can be easily reused in future projects.

Parameters **`settings`** (`ConfigObject`) – analysis settings

Returns status of the execution

Return type *StatusCode*

`eskapade.core.execution.`**`reset_eskapade`**(*skip_config=False*)

Reset Eskapade objects.

Parameters **`skip_config`** (*`bool`*) – skip reset of configuration object

eskapade.core.meta module

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

Created: 2017/09/14

Description:

A collection of (generic) meta classes for some (design) patterns:

- Singleton: Meta class for the Singleton pattern.
- Processor: Meta class with abstract methods initialize, execute, and finalize.
- ProcessorSequence: A simple (processor) sequence container.

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` `eskapade.core.meta.`**`Processor`**(*name: str*)

Bases: `object`

Processor metaclass.

`__init__`(*name: str*)

Initialize the Processor object.

`execute`()

Execution logic for processor.

`finalize`()

Finalization logic for processor.

`initialize`()

Initialization logic for processor.

`logger`

A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
```

(continues on next page)

(continued from previous page)

```
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

`name`

Get the name of processor.

Returns The name of the processor.

Return type str

`parent`

Get the group parent.

Returns The parent/group processor sequence.

`class eskapade.core.meta.`**`ProcessorSequence`**

Bases: `object`

A doubly linked processor sequence.

It remembers the order in which processors are added to the sequence. It also checks if a processor already has been added to the sequence.

`__init__``()`

Initialize the ProcessorSequence object.

add(*processor: eskapade.core.meta.Processor*) → None
Add a processor to the sequence.

Parameters **processor** (`Processor`) – The processor to add.

Raises **KeyError** – When a processor of the same type and name already exists.

clear() → None
Clear the sequence.

discard(*processor: eskapade.core.meta.Processor*) → None
Remove a processor from the sequence.

Parameters **processor** (`Processor`) – The processor to remove.

Raises **KeyError** – When the processor does not exist.

pop(*last: bool = True*) → eskapade.core.meta.Processor
Return the popped processor. Raise KeyError if empty.

By default a processor is popped from the end of the sequence.

Parameters **last** (*bool*) – Pop processor from the end of the sequence. Default is True.

Returns The pop processor.

Raises **KeyError** – When trying to pop from an empty list.

class eskapade.core.meta.**Singleton**
Bases: `type`

Metaclass for singletons.

Any instantiation of a Singleton class yields the exact same object, e.g.:

```
>>> class Klass(metaclass=Singleton):
>>>     pass
>>>
>>> a = Klass()
>>> b = Klass()
>>> a is b
True
```

See https://michaelgoerz.net/notes/singleton-objects-in-python.html.

eskapade.core.mixin module

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

Created: 2016/11/08

Classes: ArgumentsMixin, TimerMixin

Description:

Mixin classes:

- ArgumentsMixin: processes/checks arguments and sets them as attributes
- TimerMixin: keeps track of execution time
- ConfigMixin: reads and handles settings from configuration 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 `eskapade.core.mixin.`**`ArgumentsMixin`**

Bases: `object`

Mixin base class for argument parsing.

Class allows attributes to be accessed as dict items. Plus several argument processing helper functions.

`check_arg_callable`(**arg_names*, *allow_none=False*)

Check if set of arguments has iterators.

`check_arg_iters`(**arg_names*, *allow_none=False*)

Check if set of arguments has iterators.

`check_arg_opts`(*allow_none=False*, ***name_vals*)

Check if argument values are in set of options.

`check_arg_types`(*recurse=False*, *allow_none=False*, ***name_type*)

Check if set of arguments has correct types.

`check_arg_vals`(**arg_names*, *allow_none=False*)

Check if set of arguments exists as attributes and values.

`check_extra_kwargs`(*kwargs*)

Check for residual kwargs.

`check_required_args`(**arg_names*)

Check if set of arguments exists as attributes.

class `eskapade.core.mixin.`**`ConfigMixin`**(*config_path=None*)

Bases: `object`

Mixin base class for configuration settings.

`__init__`(*config_path=None*)

Initialize config settings.

Parameters **`config_path`** (`str`) – path of configuration file

`config_path`

Path of configuration file.

`get_config`(*config_path=None*)

Get settings from configuration file.

Read and return the configuration settings from a configuration file. If the path of this file is not specified as an argument, the value of the "config_path" property is used. If the file has already been read, return previous settings.

Parameters **`config_path`** (`str`) – path of configuration file

Returns configuration settings read from file

Return type configparser.ConfigParser

Raises **`RuntimeError`** – if config_path is not set

`reset_config`()

Remove previously read settings.

class `eskapade.core.mixin.`**`TimerMixin`**

Bases: `object`

Mixin base class for timing.

`__init__`()
Initialize timer.

`start_timer`()
Start run timer.

Start the timer. The timer is used to compute the run time. The returned timer start value has an undefined reference and should, therefore, only be compared to other timer values.

Returns start time in seconds

Return type float

`stop_timer`(*start_time=None*)
Stop the run timer.

Stop the timer. The timer is used to compute the run time. The elapsed time since the timer start is returned.

Parameters **`start_time`** (*`float`*) – function start_time input

Returns time difference with start in seconds

Return type float

`total_time`()
Return the total run time.

Returns total time in seconds

Return type float

eskapade.core.persistence module

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

Created: 2016/11/08

Description: Utility class and functions to get correct io path, used for persistence of results

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.

`eskapade.core.persistence.`**`create_dir`**(*dir_path*)
Create directory.

Parameters **`dir_path`** (*`str`*) – directory path

`eskapade.core.persistence.`**`io_dir`**(*io_type*, *io_conf=None*)
Construct directory path.

Parameters

- **`io_type`** (*`str`*) – type of result to store, e.g. data, macro, results.
- **`io_conf`** – IO configuration object

Returns directory path

Return type str

`eskapade.core.persistence.`**`io_path`**(*io_type*, *sub_path*, *io_conf=None*)
Construct directory path with sub path.

Parameters

- **`io_type`** (*`str`*) – type of result to store, e.g. data, macro, results.
- **`sub_path`** (*`str`*) – sub path to be included in io path
- **`io_conf`** – IO configuration object

Returns full path to directory

Return type str

`eskapade.core.persistence.`**`record_file_number`**(*file_name_base*, *file_name_ext*, *io_conf=None*)

Get next prediction-record file number.

Parameters

- **`file_name_base`** (*`str`*) – base file name
- **`file_name_ext`** (*`str`*) – file name extension
- **`io_conf`** – I/O configuration object

Returns next prediction-record file number

Return type int

`eskapade.core.persistence.`**`repl_whites`**(*name*)

Replace whitespace in names.

eskapade.core.process_manager module

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

Class: ProcessManager

Created: 2016/11/08

Description: The ProcessManager class is the heart of Eskapade. It performs initialization, execution, and finalization of analysis chains.

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` `eskapade.core.process_manager.`**`ProcessManager`**

Bases: *`eskapade.core.meta.Processor`*, *`eskapade.core.meta.ProcessorSequence`*, *`eskapade.core.mixin.TimerMixin`*

Eskapade run process manager.

ProcessManager is the event processing loop of Eskapade. It initializes, executes, and finalizes the analysis chains. There is, under normal circumstances, only one ProcessManager instance.

Here’s an simple but illustrative analysis example:

```
>>> from eskapade import process_manager, Chain, Link, StatusCode
>>>
>>> # A chain automatically registers itself with process_manager.
>>> one_plus_one_chain = Chain('one_plus_one')
>>>
>>> class OnePlusOne(Link):
>>>     def execute(self):
>>>         self.logger.info('one plus one = {result}', result=(1+1))
```

(continues on next page)

(continued from previous page)

```
>>>         return StatusCode.Success
>>>
>>> one_plus_one_chain.add(link=OnePlusOne())
>>>
>>> two_plus_two_chain = Chain('two_plus_two')
>>>
>>> class TwoPlusTwo(Link):
>>>     def execute(self):
>>>         self.logger.info('two plus two = {result}', result=(2+2))
>>>         return StatusCode.Success
>>>
>>> two_plus_two_chain.add(TwoPlusTwo())
>>>
>>> process_manager.run()
```

Ideally the user will not need to interact directly with the process manager. The magic is taken care of by the eskapade_run entry point.

`__init__`()
Initialize ProcessManager instance.

`add`(*chain: eskapade.core.element.Chain*) → None
Add a chain to the process manager.

Parameters **`chain`** (`Chain`) – The chain to add to the process manager.

Raises

- **`TypeError`** – When the chain is of an incompatible type.
- **`KeyError`** – When a chain of the same type and name already exists.

`clear`()
“Clear/remove all chains.

`execute`()
Execute all chains in order.

Returns status code of execution attempt

Return type *StatusCode*

`execute_macro`(*filename*, *copyfile=True*)
Execute an input python configuration file.

A copy of the configuration file is stored for bookkeeping purposes.

Parameters

- **`filename`** (*`str`*) – the path of the python configuration file
- **`copyfile`** (*`bool`*) – back up the macro for bookkeeping purposes

Raises **`Exception`** – if input configuration file cannot be found

`finalize`()
Finalize the process manager manager.

Returns status code of finalize attempt

Return type *StatusCode*

`get`(*chain_name: str*) → eskapade.core.element.Chain
Find the chain with the given name.

Parameters **`chain_name`** (*`str`*) – Find a chain with the given name.

Returns The chain.

Return type *Chain*

Raises **`ValueError`** – When the given chain name cannot be found.

`get_service_tree`()

Create tree of registered process-service classes.

Returns service tree

Return type dict

`get_services`()

Get set of registered process-service classes.

Returns service set

Return type set

`import_services`(*io_conf*, *chain=None*, *force=None*, *no_force=None*)

Import process services from files.

Parameters

- **`io_conf`** (*`dict`*) – I/O config as returned by ConfigObject.io_conf
- **`chain`** (*`str`*) – name of chain for which data was persisted
- **`force`** (*`bool or list`*) – force import if service already registered
- **`no_force`** (*`list`*) – do not force import of services in this list

`initialize`()

Initialize the process manager.

Initializes the process manager by configuring its chains. After initialization the configuration is printed.

Returns status code of initialize attempt

Return type *StatusCode*

`n_chains`

Return the number of chains in the process manager.

Returns The number of links in the chain.

Return type int

`persist_services`(*io_conf*, *chain=None*)

Persist process services in files.

Parameters

- **`io_conf`** (*`dict`*) – I/O config as returned by ConfigObject.io_conf
- **`chain`** (*`str`*) – name of chain for which data is persisted

`print_chains`()

Print all chains defined in the manager.

`print_services`()

Print registered process services.

`remove_all_services`()

Remove all registered process services.

remove_service (*service_cls*, *silent=False*)
Remove specified process service.

Parameters

- **service_cls** (`ProcessServiceMeta`) – service to remove
- **silent** (*bool*) – don't complain if service is not registered

reset ()
Reset the process manager.

Resetting comprises removing the chains and closing any open connections/sessions.

run () → eskapade.core.definitions.StatusCode
Run process manager.

Returns Status code of run execution.

Return type *StatusCode*

service (*service_spec*)
Get or register process service.

Parameters **service_spec** (`ProcessServiceMeta` *or* `ProcessService`) – class (instance) to register

Returns registered instance

Return type *ProcessService*

summary ()
Print process-manager summary.

Print a summary of the chains, links, and some analysis settings defined in this configuration.

eskapade.core.process_services module

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

Created: 2017/02/27

Description: Base class and core implementations of run-process services

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 eskapade.core.process_services.**ConfigObject**
Bases: *eskapade.core.process_services.ProcessService*

Configuration settings for Eskapade.

The ConfigObject is a dictionary meant for containing global settings of Eskapade. Settings are set in the configuration macro of an analysis, or on the command line.

The ConfigObject is a dictionary meant only for storing global settings of Eskapade. In general, it is accessed through the process manager.

Example usage:

```
>>> # first set logging output level.
>>> from eskapade.logger import Logger, LogLevel
>>> logger = Logger()
>>> logger.log_level = LogLevel.DEBUG
```

Obtain the ConfigObject from any location as follows:

```
>>> from eskapade import process_manager
>>> from eskapade import ConfigObject
>>> settings = process_manager.service(ConfigObject)
```

One can treat the ConfigObject as any other dictionary:

```
>>> settings['foo'] = 'bar'
>>> foo = settings['foo']
```

Write the ConfigObject to a pickle file with:

```
>>> settings.persist_in_file(file_path)
```

And reload from the pickle file with:

```
>>> settings = ConfigObject.import_from_file(file_path)
```

A ConfigObject pickle file can be read in by Eskapade with the command line option (-u).

class IoConfig(***input_config*)

Bases: `dict`

Configuration object for I/O operations.

__init__(***input_config*)

Initialize IoConfig instance.

Print()

Print a summary of the settings.

__init__()

Initialize ConfigObject instance.

add_macros(*macro_paths*)

Add configuration macros for Eskapade run.

copy()

Perform a shallow copy of self.

Returns copy

get(*setting: str*, *default: Any = None*) → object

Get value of setting. If it does not exists return the default value.

Parameters

- **setting** – The setting to get.
- **default** – The default value of the setting.

Returns The value of the setting or None if it does not exist.

io_base_dirs() → dict

Get configured base directories.

Returns base directories

Return type dict

`io_conf`()

Get I/O configuration.

The I/O configuration contains storage locations and basic analysis info.

Returns I/O configuration

Return type *IoConfig*

`set_user_opts`(*parsed_args*)

Set options specified by user on command line.

Parameters **`parsed_args`** (*`argparse.Namespace`*) – parsed user arguments

`class` `eskapade.core.process_services.`**`DataStore`**

Bases: *`eskapade.core.process_services.ProcessService`*, `dict`

Store for transient data sets and related objects.

The data store is a dictionary meant for storing transient data sets or any other objects. Links can take one or several data sets as input, transform them or use them as input for a model, and store the output back again in the datastore, to be picked up again by any following link.

Example usage:

```
>>> # first set logging output level.
>>> from eskapade.logger import Logger, LogLevel
>>> logger = Logger()
>>> logger.log_level = LogLevel.DEBUG
```

Obtain the global datastore from any location as follows:

```
>>> from eskapade import process_manager
>>> from eskapade import DataStore
>>>
>>>
>>> ds = process_manager.service(DataStore)
```

One can treat the datastore as any other dict:

```
>>> ds['a'] = 1
>>> ds['b'] = 2
>>> ds['0'] = 3
>>> a = ds['a']
```

Write the datastore to a pickle file with:

```
>>> ds.persist_in_file(file_path)
```

And reload from the pickle file with:

```
>>> ds = DataStore.import_from_file(file_path)
```

`Print`()

Print a summary the data store contents.

`class` `eskapade.core.process_services.`**`ProcessService`**

Bases: `object`

Base class for process services.

`__init__`()

Initialize service instance.

`classmethod create`()

Create an instance of this service.

Returns service instance

Return type *ProcessService*

`finish`()

Finish current processes.

This function can be implemented by a process-service implementation to finish running processes and clean up to prepare for a reset of the process manager. This would typically involve deleting large objects and closing files and database connections.

`classmethod import_from_file`(*file_path*)

Import service instance from a Pickle file.

Parameters **`file_path`** (`str`) – path of Pickle file

Returns imported service instance

Return type *ProcessService*

Raises RuntimeError, TypeError

`logger`

A logger that emits log messages to an observer.

The logger can be instantiated as a module or class attribute, e.g.

```
>>> logger = Logger()
>>> logger.info("I'm a module logger attribute.")
>>>
>>> class Point(object):
>>>     logger = Logger()
>>>
>>>     def __init__(self, x = 0.0, y = 0.0):
>>>         Point.logger.debug('Initializing {point} with x = {x}  y = {y}',␣
↪point=Point, x=x, y=y)
>>>         self._x = x
>>>         self._y = y
>>>
>>>     @property
>>>     def x(self):
>>>         self.logger.debug('Getting property x = {point._x}', point=self)
>>>         return self._x
>>>
>>>     @x.setter
>>>     def x(self, x):
>>>         self.logger.debug('Setting property y = {point._x}', point=self)
>>>         self._x = x
>>>
>>>     @property
>>>     def y(self):
>>>        self.logger.debug('Getting property y = {point._y}', point=self)
>>>        return self._y
>>>
>>>     @y.setter
>>>     def y(self, y):
```

(continues on next page)

(continued from previous page)

```
>>>         self.logger.debug('Setting property y = {point._y}', point=self)
>>>         self._y = y
>>>
>>> a_point = Point(1, 2)
>>>
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
>>> logger.log_level = LogLevel.DEBUG
>>> logger.info('p_x = {point.x} p_y = {point.y}', point=a_point)
```

The logger uses PEP-3101 (Advanced String Formatting) with named placeholders, see <https://www.python.org/dev/peps/pep-3101/> and <https://pyformat.info/> for more details and examples.

Furthermore, logging events are only formatted and evaluated for logging levels that are enabled. So, there's no need to check the logging level before logging. It's also efficient.

`persist_in_file`(*file_path*)
Persist service instance in Pickle file.

Parameters **`file_path`** (`str`) – path of Pickle file

`class` `eskapade.core.process_services.`**`ProcessServiceMeta`**
Bases: `type`

Meta class for process-services base class.

`persist`
Flag to indicate if service can be persisted.

eskapade.core.run_utils module

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

Created: 2017/04/11

Description: Utilities for Eskapade run

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.

`eskapade.core.run_utils.`**`create_arg_parser`**()
Create parser for user arguments.

An argparse parser is created and returned, ready to parse arguments specified by the user on the command line.

Returns argparse.ArgumentParser

Module contents

eskapade.core_ops package

Subpackages

eskapade.core_ops.links package

Submodules

eskapade.core_ops.links.apply module

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

Class: DsApply

Created: 2018-06-30

Description: Simple link to execute functions, to which datastore has been passed.

Helps in the development of links.

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 eskapade.core_ops.links.apply.`**`DsApply`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Simple link to execute functions to which datastore is passed.

`__init__``(`*`**kwargs`*`)`

Initialize an instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`apply`** (*`list`*) – list of functions to execute at execute(), to which datastore is passed

`execute``()`

Execute the link.

Returns status code of execution

Return type *StatusCode*

eskapade.core_ops.links.assert_in_ds module

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

Class: AssertInDs

Created: 2016/11/08

Description: Algorithm that asserts that items exists in the datastore

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 eskapade.core_ops.links.assert_in_ds.**AssertInDs**(***kwargs*)

Bases: *eskapade.core.element.Link*

Asserts that specified item(s) exists in the datastore.

__init__(***kwargs*)

Initialize link instance.

Store the configuration of link AssertInDs

Parameters

- **name** (*str*) – name of link
- **keySet** (*lst*) – list of keys to check

execute()

Execute the link.

eskapade.core_ops.links.break_link module

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

Class: Break

Created: 2017/02/26

Description: Algorithm to send break signal to process manager and halt execution

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 eskapade.core_ops.links.break_link.**Break**(***kwargs*)

Bases: *eskapade.core.element.Link*

Halt execution.

Link sends failure signal and halts execution of process manager. Break the execution of the processManager at a specific location by simply adding this link at any location in a chain.

__init__(***kwargs*)

Initialize link instance.

Parameters **name** (*str*) – name of link

execute()

Execute the link.

eskapade.core_ops.links.ds_object_deleter module

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

Class: DsObjectDeleter

Created: 2016/11/08

Description: Algorithm to delete objects from the datastore.

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 `eskapade.core_ops.links.ds_object_deleter.`**`DsObjectDeleter`**`(**kwargs)`

Bases: *`eskapade.core.element.Link`*

Delete objects from the datastore.

Delete objects from the DataStore by the key they are under, or keeps only the data by the specified keys.

`__init__``(**kwargs)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`deletion_keys`** (*`list`*) – keys to clear. Overwrites clear_all to false.
- **`deletion_classes`** (*`list`*) – delete object(s) by class type.
- **`keep_only`** (*`lsst`*) – keys to keep. Overwrites clear_all to false.
- **`clear_all`** (*`bool`*) – clear all key-value pairs in the datastore. Default is true.

`execute``()`

Execute the link.

`initialize``()`

Initialize the link.

eskapade.core_ops.links.ds_to_ds module

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

Class: DsToDs

Created: 2016/11/08

Description: Algorithm to move, copy, or remove an object in the datastore.

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 `eskapade.core_ops.links.ds_to_ds.`**`DsToDs`**`(**kwargs)`

Bases: *`eskapade.core.element.Link`*

Link to move, copy, or remove an object in the datastore.

`__init__``(**kwargs)`

Initialize link 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
- **`move`** (*`bool`*) – move read_key item to store_key. Default is true.

- **copy** (*bool*) – if True the read_key key, value pair will not be deleted. Default is false.
- **remove** (*bool*) – if True the item corresponding to read_key key will be deleted. Default is false.
- **columnsToAdd** (*dict*) – if the object is a pandas.DataFrame columns to add to the pandas.DataFrame. key = column name, value = column

execute()

Execute the link.

initialize()

Initialize the link.

eskapade.core_ops.links.event_looper module

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

Class: EventLooper

Created: 2016/11/08

Description: EventLooper algorithm processes input lines and reprints them, e.g. to use with map/reduce

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 eskapade.core_ops.links.event_looper.**EventLooper**(***kwargs*)

Bases: *eskapade.core.element.Link*

Event looper algorithm processes input lines and reprints or stores them.

Input lines are taken from sys.stdin, processed, and printed on screen.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **filename** (*str*) – file name where the strings are located (txt or similar). Default is None. (optional)
- **store_key** (*str*) – key to collect in datastore. If set lines are collected. (optional)
- **line_processor_set** (*list*) – list of functions to apply to input lines. (optional)
- **sort** (*bool*) – if true, sort lines before storage (optional)
- **unique** (*bool*) – if true, keep only unique lines before storage (optional),
- **skip_line_beginning_with** (*list*) – skip line if it starts with any of the list. input is list of strings. Default is ['#'] (optional)

execute()

Process all incoming lines.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

finalize()

Close open file if present.

initialize()
Perform basic checks of configured attributes.

eskapade.core_ops.links.hello_world module

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

Class: HelloWorld

Created: 2017/01/31

Description: Algorithm to do print Hello {}!

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 `eskapade.core_ops.links.hello_world.`**HelloWorld**(***kwargs*)
Bases: `eskapade.core.element.Link`

Defines the content of link HelloWorld.

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

Parameters

- **name** (*str*) – name assigned to the link
- **hello** (*str*) – name to print in Hello World! Defaults to ‘World’
- **repeat** (*int*) – repeat print statement N times. Default is 1

execute()
Execute the link.

eskapade.core_ops.links.import_data_store module

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

Class: ImportDataStore

Created: 2018-03-17

Description: Algorithm to import datastore from external pickle file

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 `eskapade.core_ops.links.import_data_store.`**ImportDataStore**(***kwargs*)
Bases: `eskapade.core.element.Link`

Link to import datastore from external pickle file.

Import can happen at initialize() or execute(). Default is initialize()

__init__(***kwargs*)
Initialize an instance of the datastore importer link.

Parameters

- **`name`** (*`str`*) – name of link
- **`path`** (*`str`*) – path of the datastore pickle file to import
- **`import_at_initialize`** (*`bool`*) – if false, perform datastore import at execute. Default is true, at initialize.

`execute`()
Execute the link.

Returns status code of execution

Return type *StatusCode*

`import_and_update_datastore`()
Import and update the datastore

`initialize`()
Initialize the link.

Returns status code of execution

Return type *StatusCode*

eskapade.core_ops.links.ipython_embed module

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

Class: IPythonEmbed

Created: 2017/02/26

Description: Link that starts up a python console during execution for debugging.

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 `eskapade.core_ops.links.ipython_embed.`**`IPythonEmbed`**(***kwargs*)
Bases: *`eskapade.core.element.Link`*

Link to start up a python console.

Start up a python console by simply adding this link at any location in a chain. Note: not an ipython console, but regular python console.

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

Parameters **`name`** (*`str`*) – name of link

`execute`()
Execute the link.

eskapade.core_ops.links.line_printer module

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

Class: LinePrinter

Created: 2017/02/21

Description: Simple algorithm to pick up lines and reprint them.

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 `eskapade.core_ops.links.line_printer.`**`LinePrinter`**`(`***kwargs*`)`

Bases: *`eskapade.core.element.Link`*

LinePrinter picks up lines from the datastore and prints them.

`__init__``(`***kwargs*`)`

Set up the configuration of link LinePrinter.

Parameters

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

`execute``()`

Execute the link.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

`initialize``()`

Initialize the link.

eskapade.core_ops.links.print_ds module

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

Class: PrintDs

Created: 2016/11/08

Description: Algorithm to print the content of the datastore.

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 `eskapade.core_ops.links.print_ds.`**`PrintDs`**`(`***kwargs*`)`

Bases: *`eskapade.core.element.Link`*

Print the content of the datastore.

`__init__``(`***kwargs*`)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`keys`** (*`list`*) – keys of items to print explicitly.

`execute``()`

Execute the link.

Print overview of the datastore in current state.

eskapade.core_ops.links.repeat_chain module

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

Class: RepeatChain

Created: 2016/11/08

Description: Algorithm that sends "repeat this chain" signal to processManager, until ready.

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 `eskapade.core_ops.links.repeat_chain.`**`RepeatChain`**(***kwargs*)

Bases: `eskapade.core.element.Link`

Algorithm that sends signal to processManager to repeat the current chain.

`__init__`(***kwargs*)

Link that sends signal to processManager to repeat the current chain.

Sents a RepeatChain deenums.StatusCode signal.

Parameters

- **name** (*str*) – name of link
- **listen_to** (*list*) – repeat this chain if given key is present in ConfigObject and set to true. E.g. this key is set by readtods link when looping over files.
- **maxcount** (*int*) – repeat this chain until max count has been reacher. Default is -1 (off).

`execute`()

Execute the link.

`initialize`()

Initialize the link.

eskapade.core_ops.links.skip_chain_if_empty module

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

Class: SkipChainIfEmpty

Created: 2016/11/08

Description: Algorithm to skip to the next Chain if input dataset is empty

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 `eskapade.core_ops.links.skip_chain_if_empty.`**`SkipChainIfEmpty`**(***kwargs*)

Bases: `eskapade.core.element.Link`

Sends a SkipChain enums.StatusCode signal when an appointed dataset is empty.

This signal causes that the Processs Manager to step immediately to the next Chain. Input collections can be either mongo collections or dataframes in the datastore.

`__init__`(***kwargs*)

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`collection_set`** (*`list`*) – datastore keys holding the datasets to be checked. If any of these is empty, the chain is skipped.
- **`skip_chain_when_key_not_in_ds`** (*`bool`*) – skip the chain as well if the dataframe is not present in the datastore. When True and if type is 'pandas.DataFrame', sends a SkipChain signal if key not in DataStore
- **`check_at_initialize`** (*`bool`*) – perform dataset empty is check at initialize. Default is true.
- **`check_at_execute`** (*`bool`*) – perform dataset empty is check at initialize. Default is false.

`check_collection_set``()`

Check existence of collection in either mongo or datastore, and check that they are not empty.

Collections need to be both present and not empty.

- For mongo collections a dedicated filter can be applied before doing the count.
- For pandas dataframes the additional option 'skip_chain_when_key_not_in_ds' exists. Meaning, skip the chain as well if the dataframe is not present in the datastore.

`execute``()`

Execute the link.

Skip to the next Chain if any of the input dataset is empty.

`initialize``()`

Initialize the link.

eskapade.core_ops.links.to_ds_dict module

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

Class: ToDsDict

Created: 2016/11/08

Description: Algorithm to store one object in the DataStore dict during run time.

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 `eskapade.core_ops.links.to_ds_dict.`**`ToDsDict`**`(**kwargs)`

Bases: *`eskapade.core.element.Link`*

Stores one object in the DataStore dict during run time.

`__init__``(**kwargs)`

Link to store one external object in the DataStore dict during run time.

Parameters

- **`name`** (*`str`*) – name of link
- **`store_key`** (*`str`*) – key of object to store in data store
- **`obj`** – object to store

  - **force** (*bool*) – overwrite if already present in datastore. default is false. (optional)
  - **at_initialize** (*bool*) – store at initialize of link. Default is false.
  - **at_execute** (*bool*) – store at execute of link. Default is true.
  - **copydict** (*bool*) – if true and obj is a dict, copy all key value pairs into datastore. Default is false.

do_storage(*ds*)
Perform storage in datastore.

Function makes a distinction been dicts and any other object.

execute()
Execute the link.

initialize()
Initialize the link.

Module contents

class eskapade.core_ops.links.**AssertInDs**(***kwargs*)
Bases: *eskapade.core.element.Link*

Asserts that specified item(s) exists in the datastore.

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

Store the configuration of link AssertInDs

Parameters

- **name** (*str*) – name of link
- **keySet** (*lst*) – list of keys to check

execute()
Execute the link.

class eskapade.core_ops.links.**Break**(***kwargs*)
Bases: *eskapade.core.element.Link*

Halt execution.

Link sends failure signal and halts execution of process manager. Break the execution of the processManager at a specific location by simply adding this link at any location in a chain.

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

Parameters **name** (*str*) – name of link

execute()
Execute the link.

class eskapade.core_ops.links.**DsObjectDeleter**(***kwargs*)
Bases: *eskapade.core.element.Link*

Delete objects from the datastore.

Delete objects from the DataStore by the key they are under, or keeps only the data by the specified keys.

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

Parameters

- **`name`** (*`str`*) – name of link
- **`deletion_keys`** (*`list`*) – keys to clear. Overwrites clear_all to false.
- **`deletion_classes`** (*`list`*) – delete object(s) by class type.
- **`keep_only`** (*`lsst`*) – keys to keep. Overwrites clear_all to false.
- **`clear_all`** (*`bool`*) – clear all key-value pairs in the datastore. Default is true.

`execute``()`
Execute the link.

`initialize``()`
Initialize the link.

`class` `eskapade.core_ops.links.`**`DsToDs`**`(`*`**kwargs`*`)`
Bases: *`eskapade.core.element.Link`*

Link to move, copy, or remove an object in the datastore.

`__init__``(`*`**kwargs`*`)`
Initialize link 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
- **`move`** (*`bool`*) – move read_key item to store_key. Default is true.
- **`copy`** (*`bool`*) – if True the read_key key, value pair will not be deleted. Default is false.
- **`remove`** (*`bool`*) – if True the item corresponding to read_key key will be deleted. Default is false.
- **`columnsToAdd`** (*`dict`*) – if the object is a pandas.DataFrame columns to add to the pandas.DataFrame. key = column name, value = column

`execute``()`
Execute the link.

`initialize``()`
Initialize the link.

`class` `eskapade.core_ops.links.`**`EventLooper`**`(`*`**kwargs`*`)`
Bases: *`eskapade.core.element.Link`*

Event looper algorithm processes input lines and reprints or stores them.

Input lines are taken from sys.stdin, processed, and printed on screen.

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

Parameters

- **`name`** (*`str`*) – name of link

- **filename** (*str*) – file name where the strings are located (txt or similar). Default is None. (optional)
- **store_key** (*str*) – key to collect in datastore. If set lines are collected. (optional)
- **line_processor_set** (*list*) – list of functions to apply to input lines. (optional)
- **sort** (*bool*) – if true, sort lines before storage (optional)
- **unique** (*bool*) – if true, keep only unique lines before storage (optional),
- **skip_line_beginning_with** (*list*) – skip line if it starts with any of the list. input is list of strings. Default is ['#'] (optional)

execute()

Process all incoming lines.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

finalize()

Close open file if present.

initialize()

Perform basic checks of configured attributes.

class eskapade.core_ops.links.**HelloWorld**(***kwargs*)

Bases: *eskapade.core.element.Link*

Defines the content of link HelloWorld.

__init__(***kwargs*)

Store the configuration of link HelloWorld.

Parameters

- **name** (*str*) – name assigned to the link
- **hello** (*str*) – name to print in Hello World! Defaults to 'World'
- **repeat** (*int*) – repeat print statement N times. Default is 1

execute()

Execute the link.

class eskapade.core_ops.links.**IPythonEmbed**(***kwargs*)

Bases: *eskapade.core.element.Link*

Link to start up a python console.

Start up a python console by simply adding this link at any location in a chain. Note: not an ipython console, but regular python console.

__init__(***kwargs*)

Initialize link instance.

Parameters **name** (*str*) – name of link

execute()

Execute the link.

class eskapade.core_ops.links.**LinePrinter**(***kwargs*)

Bases: *eskapade.core.element.Link*

LinePrinter picks up lines from the datastore and prints them.

`__init__`(***kwargs*)

Set up the configuration of link LinePrinter.

Parameters

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

`execute`()

Execute the link.

No output is printed except for lines that are passed on, such that the output lines can be picked up again by another parser.

`initialize`()

Initialize the link.

class `eskapade.core_ops.links.`**`PrintDs`**(***kwargs*)

Bases: *`eskapade.core.element.Link`*

Print the content of the datastore.

`__init__`(***kwargs*)

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`keys`** (*`list`*) – keys of items to print explicitly.

`execute`()

Execute the link.

Print overview of the datastore in current state.

class `eskapade.core_ops.links.`**`RepeatChain`**(***kwargs*)

Bases: *`eskapade.core.element.Link`*

Algorithm that sends signal to processManager to repeat the current chain.

`__init__`(***kwargs*)

Link that sends signal to processManager to repeat the current chain.

Sents a RepeatChain deenums.StatusCode signal.

Parameters

- **`name`** (*`str`*) – name of link
- **`listen_to`** (*`list`*) – repeat this chain if given key is present in ConfigObject and set to true. E.g. this key is set by readtods link when looping over files.
- **`maxcount`** (*`int`*) – repeat this chain until max count has been reacher. Default is -1 (off).

`execute`()

Execute the link.

`initialize`()

Initialize the link.

class `eskapade.core_ops.links.`**`SkipChainIfEmpty`**(***kwargs*)

Bases: *`eskapade.core.element.Link`*

Sends a SkipChain enums.StatusCode signal when an appointed dataset is empty.

This signal causes that the Processs Manager to step immediately to the next Chain. Input collections can be either mongo collections or dataframes in the datastore.

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

Parameters

- **`name`** (*`str`*) – name of link
- **`collection_set`** (*`list`*) – datastore keys holding the datasets to be checked. If any of these is empty, the chain is skipped.
- **`skip_chain_when_key_not_in_ds`** (*`bool`*) – skip the chain as well if the dataframe is not present in the datastore. When True and if type is 'pandas.DataFrame', sends a SkipChain signal if key not in DataStore
- **`check_at_initialize`** (*`bool`*) – perform dataset empty is check at initialize. Default is true.
- **`check_at_execute`** (*`bool`*) – perform dataset empty is check at initialize. Default is false.

`check_collection_set`()
Check existence of collection in either mongo or datastore, and check that they are not empty.

Collections need to be both present and not empty.

- For mongo collections a dedicated filter can be applied before doing the count.
- For pandas dataframes the additional option 'skip_chain_when_key_not_in_ds' exists. Meaning, skip the chain as well if the dataframe is not present in the datastore.

`execute`()
Execute the link.

Skip to the next Chain if any of the input dataset is empty.

`initialize`()
Initialize the link.

class `eskapade.core_ops.links.`**`ToDsDict`**(***kwargs*)
Bases: *`eskapade.core.element.Link`*

Stores one object in the DataStore dict during run time.

`__init__`(***kwargs*)
Link to store one external object in the DataStore dict during run time.

Parameters

- **`name`** (*`str`*) – name of link
- **`store_key`** (*`str`*) – key of object to store in data store
- **`obj`** – object to store
- **`force`** (*`bool`*) – overwrite if already present in datastore. default is false. (optional)
- **`at_initialize`** (*`bool`*) – store at initialize of link. Default is false.
- **`at_execute`** (*`bool`*) – store at execute of link. Default is true.
- **`copydict`** (*`bool`*) – if true and obj is a dict, copy all key value pairs into datastore. Default is false.

do_storage(*ds*)

Perform storage in datastore.

Function makes a distinction been dicts and any other object.

execute()

Execute the link.

initialize()

Initialize the link.

class eskapade.core_ops.links.**DsApply**(***kwargs*)

Bases: *eskapade.core.element.Link*

Simple link to execute functions to which datastore is passed.

__init__(***kwargs*)

Initialize an instance.

Parameters

- **name** (*str*) – name of link
- **apply** (*list*) – list of functions to execute at execute(), to which datastore is passed

execute()

Execute the link.

Returns status code of execution

Return type *StatusCode*

class eskapade.core_ops.links.**ImportDataStore**(***kwargs*)

Bases: *eskapade.core.element.Link*

Link to import datastore from external pickle file.

Import can happen at initialize() or execute(). Default is initialize()

__init__(***kwargs*)

Initialize an instance of the datastore importer link.

Parameters

- **name** (*str*) – name of link
- **path** (*str*) – path of the datastore pickle file to import
- **import_at_initialize** (*bool*) – if false, perform datastore import at execute. Default is true, at initialize.

execute()

Execute the link.

Returns status code of execution

Return type *StatusCode*

import_and_update_datastore()

Import and update the datastore

initialize()

Initialize the link.

Returns status code of execution

Return type *StatusCode*

Module contents

eskapade.data_quality package

Subpackages

eskapade.data_quality.links package

Submodules

eskapade.data_quality.links.fix_pandas_dataframe module

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

Class: FixPandasDataFrame

Created: 2017/04/07

Description: Link for fixing dirty pandas dataframe with inconsistent datatypes See example in: tutorials/esk501_fix_pandas_dataframe.py

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 `eskapade.data_quality.links.fix_pandas_dataframe.`**`FixPandasDataFrame`**(***kwargs*)

Bases: *`eskapade.core.element.Link`*

Fix dirty Pandas dataframe with inconsistent datatypes.

Default settings perform the following clean-up steps on an input dataframe:

- Fix all column names. E.g. remove punctuation and strange characters, and convert spaces to underscores.
- Check for various possible nans in the dataset, then make all nans consistent by turning them into numpy.nan (= float)
- Per column, assess dynamically the most consistent datatype (ignoring all nans in that column). E.g. bool, int, float, datetime64, string.
- Per column, make the data types of all rows consistent, by using the identified (or imposed) data type (by default ignoring all nans)

Boolean columns with contamination get converted to string columns by default. Optionally, they can be converted to integer columns as well.

The FixPandasDataFrame link can be used in a dataframe loop, in which case any data type assessed per column in the first dataframe iteration will be used for the next dataframes as well.

The default settings should work pretty well in many circumstances, by can be configured pretty flexibly. Optionally:

- Instead of dynamically assessed, the data type can also be imposed per column
- All nans in a column can be converted to a value consistent with the data type of that column. E.g. for integer columns, nan -> -999
- An alternative nan can be set per column and datatype
- Modifications can be applied inplace, i.e. directly to the input dataframe

`__init__`(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input data to read from data store
- **copy_columns_from_df** (*bool*) – if true, copy all columns from the dataframe (default is true)
- **original_columns** (*list*) – original (unfixed) column names to pick up from input data (required if copy_columns_from_df is set to false)
- **contaminated_columns** (*list*) – (original) columns that are known to have mistakes and that should be fixed (optional)
- **fix_column_names** (*bool*) – if true, fix column names (default is true)
- **strip_hive_prefix** (*bool*) – if true, strip table-name (hive) prefix from column names, e.g. table.bla -> bla (default is false)
- **convert_inconsistent_dtypes** (*bool*) – fix column datatypes in case of data type inconsistencies in rows (default is true)
- **var_dtype** (*dict*) – dict forcing columns to certain datatypes, e.g. {‘A’: int} (optional)
- **var_convert_inconsistent_dtypes** (*dict*) – dict allowing one to overwrite if certain columns datatypes should be fixed, e.g. {‘A’: False} (optional)
- **var_convert_func** (*dict*) – dict with datatype conversion functions for certain columns
- **check_nan_func** – boolean return function to check for nans in columns. (default is None, in which case a standard checker function gets picked up)
- **convert_inconsistent_nans** (*bool*) – if true, convert all nans to data type consistent with rest of column (default is false)
- **var_convert_inconsistent_nans** (*dict*) – dict allowing one to overwrite if certain column nans should be fixed, e.g. {‘A’: False} (optional)
- **var_nan** (*dict*) – dict with nans for certain columns (optional)
- **nan_dtype_map** (*dict*) – dictionary of nans for given data types, e.g. { int: -999 }
- **nan_default** – default nan value to which all nans found get converted (default is numpy.nan)
- **var_bool_to_int** (*list*) – convert boolean column to int (default is conversion of boolean to string)
- **inplace** (*bool*) – replace original columns; overwrites store_key to read_key (default is False)
- **store_key** (*str*) – key of output data to store in data store
- **drop_dup_rec** (*bool*) – if true, drop duplicate records from data frame after other fixes (default is false)
- **strip_string_columns** (*bool*) – if true, apply strip command to string columns (default is true)
- **cleanup_string_columns** (*list*) – boolean or list. apply cleaning-up to list of selected or all string columns. More aggressive than strip. Default is empty (= false).

execute()
Execute the link.

Fixing the Pandas dataframe consists of four steps:

- Fix all column names. E.g. remove punctuation and strange characters, and convert spaces to underscores.
- Check existing nans in that dataset, and make all nans consistent, for easy conversion later on.
- Assess most consistent datatype for each column (ignoring all nans)
- Make data types in each row consistent (by default ignoring all nans)

initialize()
Initialize the link.

`eskapade.data_quality.links.fix_pandas_dataframe.`**determine_preferred_dtype**(*dtype_cnt*)
Determine preferred column data type.

Module contents

class `eskapade.data_quality.links.`**FixPandasDataFrame**(***kwargs*)
Bases: *eskapade.core.element.Link*

Fix dirty Pandas dataframe with inconsistent datatypes.

Default settings perform the following clean-up steps on an input dataframe:

- Fix all column names. E.g. remove punctuation and strange characters, and convert spaces to underscores.
- Check for various possible nans in the dataset, then make all nans consistent by turning them into numpy.nan (= float)
- Per column, assess dynamically the most consistent datatype (ignoring all nans in that column). E.g. bool, int, float, datetime64, string.
- Per column, make the data types of all rows consistent, by using the identified (or imposed) data type (by default ignoring all nans)

Boolean columns with contamination get converted to string columns by default. Optionally, they can be converted to integer columns as well.

The FixPandasDataFrame link can be used in a dataframe loop, in which case any data type assessed per column in the first dataframe iteration will be used for the next dataframes as well.

The default settings should work pretty well in many circumstances, by can be configured pretty flexibly. Optionally:

- Instead of dynamically assessed, the data type can also be imposed per column
- All nans in a column can be converted to a value consistent with the data type of that column. E.g. for integer columns, nan -> -999
- An alternative nan can be set per column and datatype
- Modifications can be applied inplace, i.e. directly to the input dataframe

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

Parameters

- **name** (`str`) – name of link

- **`read_key`** (*`str`*) – key of input data to read from data store
- **`copy_columns_from_df`** (*`bool`*) – if true, copy all columns from the dataframe (default is true)
- **`original_columns`** (*`list`*) – original (unfixed) column names to pick up from input data (required if copy_columns_from_df is set to false)
- **`contaminated_columns`** (*`list`*) – (original) columns that are known to have mistakes and that should be fixed (optional)
- **`fix_column_names`** (*`bool`*) – if true, fix column names (default is true)
- **`strip_hive_prefix`** (*`bool`*) – if true, strip table-name (hive) prefix from column names, e.g. table.bla -> bla (default is false)
- **`convert_inconsistent_dtypes`** (*`bool`*) – fix column datatypes in case of data type inconsistencies in rows (default is true)
- **`var_dtype`** (*`dict`*) – dict forcing columns to certain datatypes, e.g. {'A': int} (optional)
- **`var_convert_inconsistent_dtypes`** (*`dict`*) – dict allowing one to overwrite if certain columns datatypes should be fixed, e.g. {'A': False} (optional)
- **`var_convert_func`** (*`dict`*) – dict with datatype conversion functions for certain columns
- **`check_nan_func`** – boolean return function to check for nans in columns. (default is None, in which case a standard checker function gets picked up)
- **`convert_inconsistent_nans`** (*`bool`*) – if true, convert all nans to data type consistent with rest of column (default is false)
- **`var_convert_inconsistent_nans`** (*`dict`*) – dict allowing one to overwrite if certain column nans should be fixed, e.g. {'A': False} (optional)
- **`var_nan`** (*`dict`*) – dict with nans for certain columns (optional)
- **`nan_dtype_map`** (*`dict`*) – dictionary of nans for given data types, e.g. { int: -999 }
- **`nan_default`** – default nan value to which all nans found get converted (default is numpy.nan)
- **`var_bool_to_int`** (*`list`*) – convert boolean column to int (default is conversion of boolean to string)
- **`inplace`** (*`bool`*) – replace original columns; overwrites store_key to read_key (default is False)
- **`store_key`** (*`str`*) – key of output data to store in data store
- **`drop_dup_rec`** (*`bool`*) – if true, drop duplicate records from data frame after other fixes (default is false)
- **`strip_string_columns`** (*`bool`*) – if true, apply strip command to string columns (default is true)
- **`cleanup_string_columns`** (*`list`*) – boolean or list. apply cleaning-up to list of selected or all string columns. More aggressive than strip. Default is empty (= false).

`execute``()`

Execute the link.

Fixing the Pandas dataframe consists of four steps:

- Fix all column names. E.g. remove punctuation and strange characters, and convert spaces to underscores.
- Check existing nans in that dataset, and make all nans consistent, for easy conversion later on.
- Assess most consistent datatype for each column (ignoring all nans)
- Make data types in each row consistent (by default ignoring all nans)

`initialize``()`

Initialize the link.

Submodules

eskapade.data_quality.dq_helper module

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

Module: data_quality.dq_helper

Created: 2017/04/11

Description: Data-quality helper functions

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.

`eskapade.data_quality.dq_helper.`**`bool_to_int`**`(`*val*, ***kwargs*`)`

Convert input boolean to int.

Parameters **`val`** – value to be evaluated

Returns evaluated value

Return type np.int64

`eskapade.data_quality.dq_helper.`**`bool_to_str`**`(`*val*, ***kwargs*`)`

Convert input boolean to str.

Parameters **`val`** – value to be evaluated

Returns evaluated value

Return type str

`eskapade.data_quality.dq_helper.`**`check_nan`**`(`*val*`)`

Check input value for not a number.

Parameters **`val`** – value to be checked for nan

Returns true if nan

Return type bool

`eskapade.data_quality.dq_helper.`**`cleanup_string`**`(`*col*`)`

Cleanup input string.

Parameters **`col`** – string to be cleaned up

Returns cleaned up string

Return type str

`eskapade.data_quality.dq_helper.`**`convert`**`(`*val*`)`

Convert input to interpreted data type.

Parameters **`val`** – value to be interpreted

Returns interpreted value

`eskapade.data_quality.dq_helper.`**`to_date_time`**`(`*val*, ***kwargs*`)`

Convert input to numpy.datetime64.

Parameters **`val`** – value to be evaluated

Returns evaluated value

Return type numpy.datetime64

`eskapade.data_quality.dq_helper.`**`to_float`**`(`*val*, ***kwargs*`)`

Convert input to float.

Parameters **`val`** – value to be evaluated

Returns evaluated value

Return type np.float64

`eskapade.data_quality.dq_helper.`**`to_int`**`(`*val*, ***kwargs*`)`

Convert input to int.

Parameters **`val`** – value to be evaluated

Returns evaluated value

Return type np.int64

`eskapade.data_quality.dq_helper.`**`to_str`**`(`*val*, ***kwargs*`)`

Convert input to string.

Parameters **`val`** – value to be converted

Returns converted value

Return type str

Module contents

eskapade.logger package

Module contents

eskapade.visualization package

Subpackages

eskapade.visualization.links package

Submodules

eskapade.visualization.links.correlation_summary module

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

Class : correlation_summary

Created: 2017/03/13

Description: Algorithm to do create correlation heatmaps.

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 `eskapade.visualization.links.correlation_summary.`**`CorrelationSummary`**(***kwargs*)

Bases: `eskapade.core.element.Link`

Create a heatmap of correlations between dataframe variables.

`__init__`(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input dataframe to read from data store
- **store_key** (*str*) – key of correlations dataframe in data store
- **results_path** (*str*) – path to save correlation summary pdf
- **methods** (*list*) – method(s) of computing correlations
- **pages_key** (*str*) – data store key of existing report pages

`execute`()

Execute the link.

`finalize`()

Finalize the link.

`initialize`()

Initialize the link.

eskapade.visualization.links.df_boxplot module

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

Class : DfBoxplot

Created: 2017/02/17

Description: Link to create a boxplot of data frame columns.

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 `eskapade.visualization.links.df_boxplot.`**`DfBoxplot`**(***kwargs*)

Bases: `eskapade.core.element.Link`

Create a boxplot of one column of a DataFrame that is grouped by values from a second column.

Creates a report page for each variable in DataFrame, containing:

- a profile of the column dataset

- a nicely scaled plot of the boxplots per group of the column

Example is available in: tutorials/esk304_df_boxplot.py

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`read_key`** (*`str`*) – key of input data to read from data store
- **`results_path`** (*`str`*) – output path of summary result files
- **`column`** (*`str`*) – column pick up from input data to use as boxplot input
- **`cause_columns`** (*`list`*) – list of columns (str) to group-by, and per unique value plot a boxplot
- **`statistics`** (*`list`*) – a list of strings of the statistics you want to generate for the boxplot the full list is taken from statistics.ArrayStats.get_latex_table defaults to: ['count', 'mean', 'min', 'max']
- **`pages_key`** (*`str`*) – data store key of existing report pages

`execute``()`

Execute the link.

Creates a report page for each column that we group-by in the data frame.

- create statistics object for group
- create overview table of column variable
- plot boxplot of column variable per group
- store plot

`finalize``()`

Finalize the link.

`initialize``()`

Initialize the link.

eskapade.visualization.links.df_summary module

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

Class : DfSummary

Created: 2017/02/17

Description: Link to create a statistics summary of data frame columns or of a set of histograms.

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 `eskapade.visualization.links.df_summary.`**`DfSummary`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Create a summary of a dataframe.

Creates a report page for each variable in data frame, containing:

- a profile of the column dataset
- a nicely scaled plot of the column dataset

Example 1 is available in: tutorials/esk301_dfsummary_plotter.py

Example 2 is available in: tutorials/esk303_histogram_filling_plotting.py Empty histograms are automatically skipped from processing.

`__init__`(***kwargs*)

Initialize link instance.

Parameters

- **`name`** (`str`) – name of link
- **`read_key`** (`str`) – key of input dataframe (or histogram-dict) to read from data store
- **`results_path`** (`str`) – output path of summary result files
- **`columns`** (`list`) – columns (or histogram keys) pick up from input data to make & plot summaries for
- **`hist_keys`** (`list`) – alternative to columns (optional)
- **`var_labels`** (`dict`) – dict of column names with a label per column
- **`var_units`** (`dict`) – dict of column names with a unit per column
- **`var_bins`** (`dict`) – dict of column names with the number of bins per column. Default per column is 30.
- **`hist_y_label`** (`str`) – y-axis label to plot for all columns. Default is ‘Bin Counts’.
- **`pages_key`** (`str`) – data store key of existing report pages

`assert_data_type`(*data*)

Check type of input data.

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

`execute`()

Execute the link.

Creates a report page for each variable in data frame.

- create statistics object for column
- create overview table of column variable
- plot histogram of column variable
- store plot

Returns execution status code

Return type *StatusCode*

`finalize`()

Finalize the link.

`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_length`(*data*)
Get length of data set.

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

Returns length of data set

`get_sample`(*data*, *key*)
Retrieve speficic column or item from input data.

Parameters

- **`data`** – input data (pandas dataframe or dict)
- **`key`** (`str`) – column key

Returns data series or item

`initialize`()
Initialize the link.

`process_1d_histogram`(*name*, *hist*)
Create statistics of and plot input 1d histogram.

Parameters

- **`name`** (`str`) – name of the histogram
- **`hist`** – input histogram object

`process_2d_histogram`(*name*, *hist*)
Create statistics of and plot input 2d histogram.

Parameters

- **`name`** (`str`) – name of the histogram
- **`hist`** – input histogram object

`process_nan_histogram`(*nphist*, *n_data*)
Process nans histogram.

Add nans histogram to pdf list

Parameters

- **`nphist`** – numpy-style input histogram, consisting of comma-separaged bin_entries, bin_edges
- **`n_data`** (`int`) – number of entries in the processed data set

`process_sample`(*name*, *sample*)
Process various possible data samples.

Parameters

- **`name`** (`str`) – name of sample
- **`sample`** – input pandas series object or histogram

`process_series`(*col*, *sample*)
Create statistics of and plot input pandas series.

Parameters

- **`col`** (`str`) – name of the series

- **`sample`** – input pandas series object

Module contents

class `eskapade.visualization.links.`**`CorrelationSummary`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Create a heatmap of correlations between dataframe variables.

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`read_key`** (*`str`*) – key of input dataframe to read from data store
- **`store_key`** (*`str`*) – key of correlations dataframe in data store
- **`results_path`** (*`str`*) – path to save correlation summary pdf
- **`methods`** (*`list`*) – method(s) of computing correlations
- **`pages_key`** (*`str`*) – data store key of existing report pages

`execute``()`

Execute the link.

`finalize``()`

Finalize the link.

`initialize``()`

Initialize the link.

class `eskapade.visualization.links.`**`DfBoxplot`**`(`*`**kwargs`*`)`

Bases: *`eskapade.core.element.Link`*

Create a boxplot of one column of a DataFrame that is grouped by values from a second column.

Creates a report page for each variable in DataFrame, containing:

- a profile of the column dataset
- a nicely scaled plot of the boxplots per group of the column

Example is available in: tutorials/esk304_df_boxplot.py

`__init__``(`*`**kwargs`*`)`

Initialize link instance.

Parameters

- **`name`** (*`str`*) – name of link
- **`read_key`** (*`str`*) – key of input data to read from data store
- **`results_path`** (*`str`*) – output path of summary result files
- **`column`** (*`str`*) – column pick up from input data to use as boxplot input
- **`cause_columns`** (*`list`*) – list of columns (str) to group-by, and per unique value plot a boxplot

- **statistics** (*list*) – a list of strings of the statistics you want to generate for the boxplot the full list is taken from statistics.ArrayStats.get_latex_table defaults to: ['count', 'mean', 'min', 'max']
- **pages_key** (*str*) – data store key of existing report pages

execute()

Execute the link.

Creates a report page for each column that we group-by in the data frame.

- create statistics object for group
- create overview table of column variable
- plot boxplot of column variable per group
- store plot

finalize()

Finalize the link.

initialize()

Initialize the link.

class eskapade.visualization.links.**DfSummary**(***kwargs*)

Bases: *eskapade.core.element.Link*

Create a summary of a dataframe.

Creates a report page for each variable in data frame, containing:

- a profile of the column dataset
- a nicely scaled plot of the column dataset

Example 1 is available in: tutorials/esk301_dfsummary_plotter.py

Example 2 is available in: tutorials/esk303_histogram_filling_plotting.py Empty histograms are automatically skipped from processing.

__init__(***kwargs*)

Initialize link instance.

Parameters

- **name** (*str*) – name of link
- **read_key** (*str*) – key of input dataframe (or histogram-dict) to read from data store
- **results_path** (*str*) – output path of summary result files
- **columns** (*list*) – columns (or histogram keys) pick up from input data to make & plot summaries for
- **hist_keys** (*list*) – alternative to columns (optional)
- **var_labels** (*dict*) – dict of column names with a label per column
- **var_units** (*dict*) – dict of column names with a unit per column
- **var_bins** (*dict*) – dict of column names with the number of bins per column. Default per column is 30.
- **hist_y_label** (*str*) – y-axis label to plot for all columns. Default is 'Bin Counts'.
- **pages_key** (*str*) – data store key of existing report pages

`assert_data_type`(*data*)
Check type of input data.

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

`execute`()
Execute the link.

Creates a report page for each variable in data frame.

- create statistics object for column
- create overview table of column variable
- plot histogram of column variable
- store plot

Returns execution status code

Return type *StatusCode*

`finalize`()
Finalize the link.

`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_length`(*data*)
Get length of data set.

Parameters `data` – input data (pandas dataframe or dict)

Returns length of data set

`get_sample`(*data*, *key*)
Retrieve speficic column or item from input data.

Parameters

- `data` – input data (pandas dataframe or dict)
- `key` (*str*) – column key

Returns data series or item

`initialize`()
Initialize the link.

`process_1d_histogram`(*name*, *hist*)
Create statistics of and plot input 1d histogram.

Parameters

- `name` (*str*) – name of the histogram
- `hist` – input histogram object

`process_2d_histogram`(*name*, *hist*)
Create statistics of and plot input 2d histogram.

Parameters

- **`name`** (*`str`*) – name of the histogram
- **`hist`** – input histogram object

`process_nan_histogram`(*nphist*, *n_data*)

Process nans histogram.

Add nans histogram to pdf list

Parameters

- **`nphist`** – numpy-style input histogram, consisting of comma-separaged bin_entries, bin_edges
- **`n_data`** (*`int`*) – number of entries in the processed data set

`process_sample`(*name*, *sample*)

Process various possible data samples.

Parameters

- **`name`** (*`str`*) – name of sample
- **`sample`** – input pandas series object or histogram

`process_series`(*col*, *sample*)

Create statistics of and plot input pandas series.

Parameters

- **`col`** (*`str`*) – name of the series
- **`sample`** – input pandas series object

Submodules

eskapade.visualization.vis_utils module

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

Created: 2017/02/28

Description: Utility functions to collect Eskapade python modules e.g. functions to get correct Eskapade file paths and env variables

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.

`eskapade.visualization.vis_utils.`**`box_plot`**(*df*, *cause_col*, *result_col='cost'*, *pdf_file_name=''*, *ylim_quant=0.95*, *ylim_high=None*, *ylim_low=0*, *rot=90*, *statlim=400*, *label_dict=None*, *title_add=''*, *top=20*)

Make box plot.

Function that plots the boxplot of the column df[result_col] in groups of cause_col. This means that the DataFrame is grouped-by on the cause column and then the distribution per group is plotted in a boxplot using the standard pandas functionality. Boxplots with less than statlim (default=400) entries in it are automatically removed.

Parameters

- **`df`** – pandas DataFrame
- **`cause_col`** (*`str`*) – name of the column to group on. This can technically be a number, but that is uncommon.
- **`result_col`** (*`str`*) – column to do the boxplot on
- **`pdf_file_name`** (*`str`*) – if set, will store the plot in a pdf file
- **`ylim_quant`** (*`float`*) – the quantile of the y upper limit
- **`ylim_high`** (*`float`*) – when defined, this limit is used, when not defined, defaults to None and ylim_high is determined by ylim_quant
- **`ylim_low`** (*`float`*) – matplotlib set_ylim lower bound
- **`rot`** (*`int`*) – matplotlib rot
- **`statlim`** (*`int`*) – the number of entries that a group is required to have in order to be plotted
- **`label_dict`** (*`dict`*) – dictionary with labels for the columns, usage example: label_dict={'col_x': 'Time'}
- **`title_add`** (*`str`*) – string that is added to the automatic title (the y column name)
- **`top`** (*`int`*) – only print the top 20 characters of x-labels and y-labels. (default is 20)

`eskapade.visualization.vis_utils.`**`delete_smallstat`**(*df*, *group_col*, *statlim=400*)

Remove low-statistics groups from dataframe.

Function to make a new DataFrame that removes all groups of group_col that have less than statlim entries.

Parameters

- **`df`** – pandas DataFrame
- **`group_col`** (*`str`*) – name of the column to group on
- **`statlim`** (*`int`*) – number of entries a group has to have to be statistically significant

Returns smaller DataFrame and the number of removed categories

Return type tuple

`eskapade.visualization.vis_utils.`**`plot_2d_histogram`**(*hist*, *x_lim*, *y_lim*, *title*, *x_label*, *y_label*, *pdf_file_name*)

Plot 2d histogram with matplotlib.

Parameters

- **`hist`** – input numpy histogram = x_bin_edges, y_bin_edges, bin_entries_2dgrid
- **`x_lim`** (*`tuple`*) – range tuple of x-axis (min,max)
- **`y_lim`** (*`tuple`*) – range tuple of y-axis (min,max)
- **`title`** (*`str`*) – title of plot
- **`x_label`** (*`str`*) – Label for histogram x-axis
- **`y_label`** (*`str`*) – Label for histogram y-axis
- **`pdf_file_name`** (*`str`*) – if set, will store the plot in a pdf file

eskapade.visualization.vis_utils.**plot_correlation_matrix**(*matrix_colors*, *x_labels*, *y_labels*, *pdf_file_name=''*, *title='correlation'*, *vmin=-1*, *vmax=1*, *color_map='RdYlGn'*, *x_label=''*, *y_label=''*, *top=20*, *matrix_numbers=None*, *print_both_numbers=True*)

Create and plot correlation matrix.

Parameters

- **matrix_colors** – input correlation matrix
- **x_labels** (*list*) – Labels for histogram x-axis bins
- **y_labels** (*list*) – Labels for histogram y-axis bins
- **pdf_file_name** (*str*) – if set, will store the plot in a pdf file
- **title** (*str*) – if set, title of the plot
- **vmin** (*float*) – minimum value of color legend (default is -1)
- **vmax** (*float*) – maximum value of color legend (default is +1)
- **x_label** (*str*) – Label for histogram x-axis
- **y_label** (*str*) – Label for histogram y-axis
- **color_map** (*str*) – color map passed to matplotlib pcolormesh. (default is 'RdYlGn')
- **top** (*int*) – only print the top 20 characters of x-labels and y-labels. (default is 20)
- **matrix_numbers** – input matrix used for plotting numbers. (default it matrix_colors)

eskapade.visualization.vis_utils.**plot_histogram**(*hist*, *x_label*, *y_label=None*, *is_num=True*, *is_ts=False*, *pdf_file_name=''*, *top=20*)

Create and plot histogram of column values.

Parameters

- **hist** – input numpy histogram = values, bin_edges
- **x_label** (*str*) – Label for histogram x-axis
- **y_label** (*str*) – Label for histogram y-axis
- **is_num** (*bool*) – True if observable to plot is numeric
- **is_ts** (*bool*) – True if observable to plot is a timestamp
- **pdf_file_name** (*str*) – if set, will store the plot in a pdf file
- **top** (*int*) – only print the top 20 characters of x-labels and y-labels. (default is 20)

Module contents

Submodules

eskapade.entry_points module

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

Created: 2017-08-08

Description: Collection of eskapade entry points

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.

`eskapade.entry_points.`**`eskapade_bootstrap`**`()`
Generate Eskapade project structure.

`eskapade.entry_points.`**`eskapade_generate_link`**`()`
Generate Eskapade link.

By default does not create init file.

`eskapade.entry_points.`**`eskapade_generate_macro`**`()`
Generate Eskapade macro.

`eskapade.entry_points.`**`eskapade_generate_notebook`**`()`
Generate Eskapade notebook.

`eskapade.entry_points.`**`eskapade_ignite`**`()`
Log info message.

`eskapade.entry_points.`**`eskapade_run`**`()`
Run Eskapade.

Top-level entry point for an Eskapade run started from the command line. Arguments specified by the user are parsed and converted to settings in the configuration object. Optionally, an interactive Python session is started when the run is finished.

`eskapade.entry_points.`**`eskapade_trial`**`()`
Run Eskapade tests.

We will keep this here until we've completed switch to pytest or nose and tox. We could also keep it, but I don't like the fact that packages etc. are hard coded. Gotta come up with a better solution.

eskapade.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

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

exception `eskapade.exceptions.`**`MissingPackageError`**(*message=”*, *required_by=”*)

Bases: `Exception`

Exception raised if third-party package 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

eskapade.helpers module

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

Module: helpers

Created: 2017/05/17

Description: Helper functions for Eskapade

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.

`eskapade.helpers.`**`apply_transform_funcs`**(*obj*, *trans_funcs*, *func_args=None*, *func_kwargs=None*)

Transform object by applying transformation functions.

Transformation functions are applied sequentially to the output of the previous function, starting with the specified object. The final resulting object is returned.

Functions are specified with the “trans_funcs” argument. This is an iterable of either the functions themselves or (function, positional arguments, keyword arguments) combinations. The function arguments can also be specified by the “func_args” and “func_kwargs” arguments. The functions are called with the object to be transformed as a first argument, followed by the specified positional and keyword arguments.

A specified function can be either a callable object or a string. The latter is interpreted as a method of the type of the object on which the function is applied.

Parameters

- **`obj`** – object to transform
- **`trans_funcs`** (*`iterable`*) – functions to apply, specified the function or a (function, args, kwargs) tuple
- **`func_args`** (*`dict`*) – function positional arguments, specified as (function, arguments tuple) pairs
- **`func_kwargs`** (*`dict`*) – function keyword arguments, specified as (function, kwargs dict) pairs

Returns transformed object

`eskapade.helpers.`**`obj_repr`**(*obj*)

Get generic string representation of object.

`eskapade.helpers.`**`process_transform_funcs`**(*trans_funcs*, *func_args=None*, *func_kwargs=None*)

Process input of the apply_transform_funcs function.

Parameters

- **`trans_funcs`** (*`iterable`*) – functions to apply, specified the function or a (function, args, kwargs) tuple
- **`func_args`** (*`dict`*) – function positional arguments, specified as (function, arguments tuple) pairs
- **`func_kwargs`** (*`dict`*) – function keyword arguments, specified as (function, kwargs dict) pairs

Returns transformation functions for apply_transform_funcs function

Return type list

eskapade.resources module

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

Created: 2017/08/23

Description: Collection of helper functions to get fixtures, i.e. test data, ROOT/RooFit libs, and tutorials. These are mostly used by the (integration) tests.

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.

`eskapade.resources.`**`config`**(*name: str*) → str

Return the absolute path of a config.

Parameters **`name`** (*`str`*) – The name of the config.

Returns The absolute path of the config.

Raises **`FileNotFoundError`** – If the config cannot be found.

`eskapade.resources.`**`fixture`**(*name: str*) → str

Return the full path filename of a fixture data set.

Parameters **`name`** (*`str`*) – The name of the fixture.

Returns The full path filename of the fixture data set.

Return type str

Raises **`FileNotFoundError`** – If the fixture cannot be found.

`eskapade.resources.`**`lib`**(*name: str*) → str

Return the full path filename of a library.

Parameters **`name`** (*`str`*) – The name of the library.

Returns The full path filename of the library.

Return type str

Raises **`FileNotFoundError`** – If the library cannot be found.

`eskapade.resources.`**`template`**(*name: str*) → str

Return the full path filename of a tutorial.

Parameters **`name`** (*`str`*) – The name of the template.

Returns The full path filename of the tutorial.

Return type str

Raises **`FileNotFoundError`** – If the template cannot be found.

`eskapade.resources.`**`tutorial`**(*name: str*) → str

Return the full path filename of a tutorial.

Parameters **`name`** (*`str`*) – The name of the tutorial.

Returns The full path filename of the tutorial.

Return type str

Raises **`FileNotFoundError`** – If the tutorial cannot be found.

eskapade.utils module

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

Created: 2016/11/08

Description: Utility functions to collect Eskapade python modules e.g. functions to get correct Eskapade file paths and env variables

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.

`eskapade.utils.`**`collect_python_modules`**()

Collect Eskapade Python modules.

`eskapade.utils.`**`get_env_var`**(*key*)

Retrieve Eskapade-specific environment variables.

Parameters **`key`** (*`str`*) – Eskapade-specific key to variable

Returns environment variable value

Return type str

`eskapade.utils.`**`set_matplotlib_backend`**(*backend=None*, *batch=None*, *silent=True*)

Set Matplotlib backend.

Parameters

- **`backend`** (*`str`*) – backend to set
- **`batch`** (*`bool`*) – require backend to be non-interactive
- **`silent`** (*`bool`*) – do not raise exception if backend cannot be set

Raises RuntimeError

eskapade.version module

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

Module contents

6.10 Appendices

6.10.1 Miscellaneous

Collection of miscelleneous Eskapade related items.

- See *Migration Tips* to migrate between Eskapade versions.
- See *macOS* to get started with Eskapade on a mac.

Migration Tips

From version 0.6 to 0.7

Below we list the API changes needed to migrate from Eskapade version 0.6 to version 0.7.

Links

- Process manager definition:
  - `proc_mgr.` change to `process_manager.`
  - `ProcessManager` change to `process_manager`
  - Delete line: `proc_mgr = ProcessManager()`
- Logger:
  - Change `log().` to `logger.`

Macros

- Process manager definition:
  - `proc_mgr.` change to `process_manager.`
  - `ProcessManager` change to `process_manager`
  - Delete line: `proc_mgr = ProcessManager()`
- Logger:
  - `import logging` change to `from eskapade.logger import Logger, LogLevel`
  - `log.` change to `logger.`
  - `log = logging.getLogger('macro.cpf_analysis')` change to `logger = Logger()`
  - `logging` change to `LogLevel`
- Settings:

  Remove `os.environ['WORKDIRROOT']`, since the environment variable WORKDIRROOT is no longer defined, define explicitly the data and macro paths, or execute the macros and tests from the root directory of the project, resulting in something like:

  - `settings['resultsDir'] = os.getcwd() + 'es_results'`
  - `settings['macrosDir'] = os.getcwd() + 'es_macros'`
  - `settings['dataDir'] = os.getcwd() + 'data'`
- Chain definition in macros:
  - To import the Chain object add `from eskapade import Chain`
  - Change `process_manager.add_chain('chain_name')` to `<chain_name> = Chain('chain_name')`
  - `process_manager.get_chain('ReadCSV').add_link` to `<chain_name>.add`

Tests

- Process manager definition:
  - Change `ProcessManager()` to `process_manager`
  - Change `process_manager.get_chain` to `process_manager.get`
- Settings:

  Remove `os.environ['WORKDIRROOT']`, since the environment variable WORKDIRROOT is no longer defined, define explicitly the data and macro paths, or execute the macros and tests from the root directory of the project, resulting in something like:
  - `settings['macrosDir'] = os.getcwd() + '/es_macros'`
  - `settings['dataDir'] = os.getcwd() + '/data'`
- StatusCode:
  - Change `status.isSkipChain()` to `status.is_skip_chain()`

macOS

To install Eskapade on macOS there are basically four steps:

- Setting up Python 3.6
- Setting up Apache Spark 2.x
- Setting up ROOT 6.10.08
- Setting up Eskapade

Note: This installation guide is written for macOS High Sierra with Homebrew, and fish.

Setting up Python 3.6

Homebrew provides Python 3.6 for macOS:

```
$ brew install python3
```

To create an isolated Python installation use `virtualenv`:

```
$ virtualenv venv/eskapade --python=python3 --system-site-packages
```

Each time a new terminal is started, set up the virtual python environment:

```
$ . ~/venv/eskapade/bin/activate.fish
```

Setting up ROOT 6

Clone ROOT from the git repository:

```
git clone http://root.cern.ch/git/root.git
cd root
git checkout -b v6-10-08 v6-10-08
```

Then compile it with the additional flags to ensure the desired functionality:

```
$ mkdir ~/root_v06-10-08_p36m && cd ~/root_v06-10-08_p36m
$ cmake -Dfftw3=ON -Dmathmore=ON -Dminuit2=ON -Droofit=ON -Dtmva=ON -Dsoversion=ON -
↪Dthread=ON -Dpython3=ON -DPYTHON_EXECUTABLE=/usr/local/opt/python3/Frameworks/
↪Python.framework/Versions/3.6/bin/python3.6m -DPYTHON_INCLUDE_DIR=/usr/local/opt/
↪python3/Frameworks/Python.framework/Versions/3.6/include/python3.6m/ -DPYTHON_
↪LIBRARY=/usr/local/opt/python3/Frameworks/Python.framework/Versions/3.6/lib/
↪libpython3.6m.dylib $HOME/root
$ cmake --build . -- -j7
```

PS: make sure all the flags are picked up correctly (for example, `-Dfftw3` requires `fftw` to be installed with Homebrew).

To setup the ROOT environment each time a new shell is started, set the following environment variables:

```
set -xg ROOTSYS "$HOME/root_v06-10-08_p36m"
set -xg PATH $ROOTSYS/bin $PATH
set -xg LD_LIBRARY_PATH "$ROOTSYS/lib:$LD_LIBRARY_PATH"
set -xg DYLD_LIBRARY_PATH "$ROOTSYS/lib:$DYLD_LIBRARY_PATH"
set -xg LIBPATH "$ROOTSYS/lib:$LIBPATH"
set -xg SHLIB_PATH "$ROOTSYS/lib:$SHLIB_PATH"
set -xg PYTHONPATH "$ROOTSYS/lib:$PYTHONPATH"
```

Note that for bash shells this can be done by sourcing the script in `root_v06-10-08_p36m/bin/thisroot.sh`.

Finally, install the Python packages for ROOT bindings:

```
$ pip install rootpy==1.0.1 root-numpy=4.7.3
```

Setting up Apache Spark 2.x

Apache Spark is provided through Homebrew:

```
$ brew install apache-spark
```

The `py4j` package is needed to support access to Java objects from Python:

```
$ pip install py4j==0.10.4
```

To set up the Spark environment each time a new terminal is started set:

```
set -xg SPARK_HOME (brew --prefix apache-spark)/libexec
set -xg SPARK_LOCAL_HOSTNAME "localhost"
set -xg PYTHONPATH "$SPARK_HOME/python:$PYTHONPATH"
```

Setting up Eskapade

The Eskapade source code can be obtained from git:

```
$ git clone git@github.com:KaveIO/Eskapade.git eskapade
```

To set up the Eskapade environment (Python, Spark, ROOT) each time a new terminal is started, source a shell script (e.g. `setup_eskapade.fish`) that contains set the environment variables as described above:

```
# --- setup Python
. ~/venv/eskapade/bin/activate.fish

# --- setup ROOT
set -xg ROOTSYS "${HOME}/root_v06-10-08_p36m"
set -xg PATH $ROOTSYS/bin $PATH
set -xg LD_LIBRARY_PATH "$ROOTSYS/lib:$LD_LIBRARY_PATH"
set -xg DYLD_LIBRARY_PATH "$ROOTSYS/lib:$DYLD_LIBRARY_PATH"
set -xg LIBPATH "$ROOTSYS/lib:$LIBPATH"
set -xg SHLIB_PATH "$ROOTSYS/lib:$SHLIB_PATH"
set -xg PYTHONPATH "$ROOTSYS/lib:$PYTHONPATH"

# --- setup Spark
set -xg SPARK_HOME (brew --prefix apache-spark)/libexec
set -xg SPARK_LOCAL_HOSTNAME "localhost"
set -xg PYTHONPATH "$SPARK_HOME/python:$PYTHONPATH"

# --- setup Eskapade
cd /path/to/eskapade
```

Finally, install Eskapade (and it's dependencies) by simply running:

```
$ pip install -e /path/to/eskapade
```

6.11 Indices and tables

- genindex
- modindex

Python Module Index

e

eskapade, 140
eskapade.analysis, 86
eskapade.analysis.correlation, 69
eskapade.analysis.datetime, 69
eskapade.analysis.histogram, 71
eskapade.analysis.histogram_filling, 80
eskapade.analysis.links, 60
eskapade.analysis.links.apply_func_to_df, 47
eskapade.analysis.links.apply_selection_to_df, 48
eskapade.analysis.links.basic_generator, 49
eskapade.analysis.links.df_concatenator, 50
eskapade.analysis.links.df_merger, 50
eskapade.analysis.links.histogrammar_filler, 51
eskapade.analysis.links.random_sample_splitter, 53
eskapade.analysis.links.read_to_df, 54
eskapade.analysis.links.record_factorizer, 55
eskapade.analysis.links.record_vectorizer, 56
eskapade.analysis.links.value_counter, 57
eskapade.analysis.links.write_from_df, 59
eskapade.analysis.statistics, 83
eskapade.core, 105
eskapade.core.definitions, 86
eskapade.core.element, 88
eskapade.core.exceptions, 91
eskapade.core.execution, 91
eskapade.core.meta, 92
eskapade.core.mixin, 94
eskapade.core.persistence, 96
eskapade.core.process_manager, 97
eskapade.core.process_services, 100
eskapade.core.run_utils, 104
eskapade.core_ops, 120
eskapade.core_ops.links, 114
eskapade.core_ops.links.apply, 105
eskapade.core_ops.links.assert_in_ds, 105
eskapade.core_ops.links.break_link, 106
eskapade.core_ops.links.ds_object_deleter, 106
eskapade.core_ops.links.ds_to_ds, 107
eskapade.core_ops.links.event_looper, 108
eskapade.core_ops.links.hello_world, 109
eskapade.core_ops.links.import_data_store, 109
eskapade.core_ops.links.ipython_embed, 110
eskapade.core_ops.links.line_printer, 110
eskapade.core_ops.links.print_ds, 111
eskapade.core_ops.links.repeat_chain, 112
eskapade.core_ops.links.skip_chain_if_empty, 112
eskapade.core_ops.links.to_ds_dict, 113
eskapade.data_quality, 125
eskapade.data_quality.dq_helper, 124
eskapade.data_quality.links, 122
eskapade.data_quality.links.fix_pandas_dataframe, 120
eskapade.entry_points, 136
eskapade.exceptions, 136
eskapade.helpers, 137
eskapade.logger, 125
eskapade.resources, 138
eskapade.utils, 139
eskapade.version, 139
eskapade.visualization, 136

`eskapade.visualization.links`, 130
`eskapade.visualization.links.correlation_summary`, 125
`eskapade.visualization.links.df_boxplot`, 126
`eskapade.visualization.links.df_summary`, 127
`eskapade.visualization.vis_utils`, 133

Index

Symbols

__init__() (eskapade.analysis.datetime.FreqTimePeriod method), 69
__init__() (eskapade.analysis.datetime.TimePeriod method), 70
__init__() (eskapade.analysis.datetime.UniformTsTimePeriod method), 71
__init__() (eskapade.analysis.histogram.BinningUtil method), 72
__init__() (eskapade.analysis.histogram.Histogram method), 73
__init__() (eskapade.analysis.histogram.ValueCounts method), 78
__init__() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 80
__init__() (eskapade.analysis.links.ApplyFuncToDf method), 60
__init__() (eskapade.analysis.links.ApplySelectionToDf method), 60
__init__() (eskapade.analysis.links.BasicGenerator method), 61
__init__() (eskapade.analysis.links.DfConcatenator method), 61
__init__() (eskapade.analysis.links.DfMerger method), 62
__init__() (eskapade.analysis.links.HistogrammarFiller method), 62
__init__() (eskapade.analysis.links.RandomSampleSplitter method), 64
__init__() (eskapade.analysis.links.ReadToDf method), 64
__init__() (eskapade.analysis.links.RecordFactorizer method), 65
__init__() (eskapade.analysis.links.RecordVectorizer method), 66
__init__() (eskapade.analysis.links.ValueCounter method), 67
__init__() (eskapade.analysis.links.WriteFromDf method), 68
__init__() (eskapade.analysis.links.apply_func_to_df.ApplyFuncToDf method), 47
__init__() (eskapade.analysis.links.apply_selection_to_df.ApplySelectionT method), 48
__init__() (eskapade.analysis.links.basic_generator.BasicGenerator method), 49
__init__() (eskapade.analysis.links.df_concatenator.DfConcatenator method), 50
__init__() (eskapade.analysis.links.df_merger.DfMerger method), 50
__init__() (eskapade.analysis.links.histogrammar_filler.HistogrammarFiller method), 51
__init__() (eskapade.analysis.links.random_sample_splitter.RandomSample method), 53
__init__() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
__init__() (eskapade.analysis.links.record_factorizer.RecordFactorizer method), 55
__init__() (eskapade.analysis.links.record_vectorizer.RecordVectorizer method), 56
__init__() (eskapade.analysis.links.value_counter.ValueCounter method), 57
__init__() (eskapade.analysis.links.write_from_df.WriteFromDf method), 59
__init__() (eskapade.analysis.statistics.ArrayStats method), 83
__init__() (eskapade.analysis.statistics.GroupByStats method), 85
__init__() (eskapade.core.definitions.RandomSeeds method), 87
__init__() (eskapade.core.element.Chain method), 88
__init__() (eskapade.core.element.Link method), 90
__init__() (eskapade.core.meta.Processor method), 92
__init__() (eskapade.core.meta.ProcessorSequence method), 93
__init__() (eskapade.core.mixin.ConfigMixin method), 95
__init__() (eskapade.core.mixin.TimerMixin method), 95
__init__() (eskapade.core.process_manager.ProcessManager method), 98

__init__() (eskapade.core.process_services.ConfigObject method), 101
__init__() (eskapade.core.process_services.ConfigObject.IoConfig method), 101
__init__() (eskapade.core.process_services.ProcessService method), 102
__init__() (eskapade.core_ops.links.AssertInDs method), 114
__init__() (eskapade.core_ops.links.Break method), 114
__init__() (eskapade.core_ops.links.DsApply method), 119
__init__() (eskapade.core_ops.links.DsObjectDeleter method), 114
__init__() (eskapade.core_ops.links.DsToDs method), 115
__init__() (eskapade.core_ops.links.EventLooper method), 115
__init__() (eskapade.core_ops.links.HelloWorld method), 116
__init__() (eskapade.core_ops.links.IPythonEmbed method), 116
__init__() (eskapade.core_ops.links.ImportDataStore method), 119
__init__() (eskapade.core_ops.links.LinePrinter method), 116
__init__() (eskapade.core_ops.links.PrintDs method), 117
__init__() (eskapade.core_ops.links.RepeatChain method), 117
__init__() (eskapade.core_ops.links.SkipChainIfEmpty method), 118
__init__() (eskapade.core_ops.links.ToDsDict method), 118
__init__() (eskapade.core_ops.links.apply.DsApply method), 105
__init__() (eskapade.core_ops.links.assert_in_ds.AssertInDs method), 106
__init__() (eskapade.core_ops.links.break_link.Break method), 106
__init__() (eskapade.core_ops.links.ds_object_deleter.DsObjectDeleter method), 107
__init__() (eskapade.core_ops.links.ds_to_ds.DsToDs method), 107
__init__() (eskapade.core_ops.links.event_looper.EventLooper method), 108
__init__() (eskapade.core_ops.links.hello_world.HelloWorld method), 109
__init__() (eskapade.core_ops.links.import_data_store.ImportDataStore method), 109
__init__() (eskapade.core_ops.links.ipython_embed.IPythonEmbed method), 110
__init__() (eskapade.core_ops.links.line_printer.LinePrinter method), 111
__init__() (eskapade.core_ops.links.print_ds.PrintDs method), 111
__init__() (eskapade.core_ops.links.repeat_chain.RepeatChain method), 112
__init__() (eskapade.core_ops.links.skip_chain_if_empty.SkipChainIfEmpt method), 112
__init__() (eskapade.core_ops.links.to_ds_dict.ToDsDict method), 113
__init__() (eskapade.data_quality.links.FixPandasDataFrame method), 122
__init__() (eskapade.data_quality.links.fix_pandas_dataframe.FixPandasDa method), 120
__init__() (eskapade.exceptions.MissingPackageError method), 137
__init__() (eskapade.visualization.links.CorrelationSummary method), 130
__init__() (eskapade.visualization.links.DfBoxplot method), 130
__init__() (eskapade.visualization.links.DfSummary method), 131
__init__() (eskapade.visualization.links.correlation_summary.CorrelationSu method), 126
__init__() (eskapade.visualization.links.df_boxplot.DfBoxplot method), 127
__init__() (eskapade.visualization.links.df_summary.DfSummary method), 128

A

add() (eskapade.core.element.Chain method), 88
add() (eskapade.core.meta.ProcessorSequence method), 93
add() (eskapade.core.process_manager.ProcessManager method), 98
add_apply_func() (eskapade.analysis.links.apply_func_to_df.ApplyFuncToDf method), 48
add_apply_func() (eskapade.analysis.links.ApplyFuncToDf method), 60
add_macros() (eskapade.core.process_services.ConfigObject method), 101
apply_transform_funcs() (in module eskapade.helpers), 137
ApplyFuncToDf (class in eskapade.analysis.links), 60
ApplyFuncToDf (class in eskapade.analysis.links.apply_func_to_df), 47
ApplySelectionToDf (class in eskapade.analysis.links), 60
ApplySelectionToDf (class in eskapade.analysis.links.apply_selection_to_df), 48
ArgumentsMixin (class in eskapade.core.mixin), 95
ArrayStats (class in eskapade.analysis.statistics), 83
assert_data_type() (eskapade.visualization.links.df_summary.DfSummary

method), 128
assert_data_type() (eskapade.visualization.links.DfSummary method), 131
assert_dataframe() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
AssertInDs (class in eskapade.core_ops.links), 114
AssertInDs (class in eskapade.core_ops.links.assert_in_ds), 106

B

BasicGenerator (class in eskapade.analysis.links), 61
BasicGenerator (class in eskapade.analysis.links.basic_generator), 49
bin_centers() (eskapade.analysis.histogram.Histogram method), 74
bin_edges() (eskapade.analysis.histogram.Histogram method), 74
bin_entries() (eskapade.analysis.histogram.Histogram method), 74
bin_labels() (eskapade.analysis.histogram.Histogram method), 74
bin_specs (eskapade.analysis.histogram.BinningUtil attribute), 72
BinningUtil (class in eskapade.analysis.histogram), 71
bool_to_int() (in module eskapade.data_quality.dq_helper), 124
bool_to_str() (in module eskapade.data_quality.dq_helper), 124
box_plot() (in module eskapade.visualization.vis_utils), 133
Break (class in eskapade.core_ops.links), 114
Break (class in eskapade.core_ops.links.break_link), 106
BreakChain (eskapade.core.definitions.StatusCode attribute), 87

C

categorize_columns() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
Chain (class in eskapade.core.element), 88
check_arg_callable() (eskapade.core.mixin.ArgumentsMixin method), 95
check_arg_iters() (eskapade.core.mixin.ArgumentsMixin method), 95
check_arg_opts() (eskapade.core.mixin.ArgumentsMixin method), 95
check_arg_types() (eskapade.core.mixin.ArgumentsMixin method), 95
check_arg_vals() (eskapade.core.mixin.ArgumentsMixin method), 95
check_collection_set() (eskapade.core_ops.links.skip_chain_if_empty.SkipChainIfEmpty method), 113
check_collection_set() (eskapade.core_ops.links.SkipChainIfEmpty method), 118
check_extra_kwargs() (eskapade.core.mixin.ArgumentsMixin method), 95
check_nan() (in module eskapade.data_quality.dq_helper), 124
check_required_args() (eskapade.core.mixin.ArgumentsMixin method), 95
chi2_from_rho() (in module eskapade.analysis.correlation), 69
cleanup_string() (in module eskapade.data_quality.dq_helper), 124
clear() (eskapade.core.element.Chain method), 88
clear() (eskapade.core.meta.ProcessorSequence method), 94
clear() (eskapade.core.process_manager.ProcessManager method), 98
collect_python_modules() (in module eskapade.utils), 139
combine_hists() (eskapade.analysis.histogram.Histogram class method), 74
config() (in module eskapade.resources), 138
config_path (eskapade.core.mixin.ConfigMixin attribute), 95
ConfigMixin (class in eskapade.core.mixin), 95
ConfigObject (class in eskapade.core.process_services), 100
ConfigObject.IoConfig (class in eskapade.core.process_services), 101
construct_empty_hist() (eskapade.analysis.links.histogrammar_filler.HistogrammarFiller method), 52
construct_empty_hist() (eskapade.analysis.links.HistogrammarFiller method), 63
convert() (in module eskapade.data_quality.dq_helper), 124
copy() (eskapade.analysis.histogram.Histogram method), 74
copy() (eskapade.core.process_services.ConfigObject method), 101
CorrelationSummary (class in eskapade.visualization.links), 130
CorrelationSummary (class in eskapade.visualization.links.correlation_summary), 126
count() (eskapade.analysis.histogram.ValueCounts method), 78

counts (eskapade.analysis.histogram.ValueCounts attribute), 78
create() (eskapade.core.process_services.ProcessService class method), 103
create_arg_parser() (in module eskapade.core.run_utils), 104
create_dir() (in module eskapade.core.persistence), 96
create_mpv_stat() (eskapade.analysis.statistics.ArrayStats method), 83
create_stats() (eskapade.analysis.statistics.ArrayStats method), 84
create_sub_counts() (eskapade.analysis.histogram.ValueCounts method), 78

D

DataStore (class in eskapade.core.process_services), 102
datatype (eskapade.analysis.histogram.Histogram attribute), 74
delete_smallstat() (in module eskapade.visualization.vis_utils), 134
determine_preferred_dtype() (in module eskapade.data_quality.links.fix_pandas_dataframe), 122
DfBoxplot (class in eskapade.visualization.links), 130
DfBoxplot (class in eskapade.visualization.links.df_boxplot), 126
DfConcatenator (class in eskapade.analysis.links), 61
DfConcatenator (class in eskapade.analysis.links.df_concatenator), 50
DfMerger (class in eskapade.analysis.links), 62
DfMerger (class in eskapade.analysis.links.df_merger), 50
DfSummary (class in eskapade.visualization.links), 131
DfSummary (class in eskapade.visualization.links.df_summary), 127
discard() (eskapade.core.element.Chain method), 88
discard() (eskapade.core.meta.ProcessorSequence method), 94
do_storage() (eskapade.core_ops.links.to_ds_dict.ToDsDict method), 114
do_storage() (eskapade.core_ops.links.ToDsDict method), 118
drop_inconsistent_keys() (eskapade.analysis.links.value_counter.ValueCounter method), 58
drop_inconsistent_keys() (eskapade.analysis.links.ValueCounter method), 67
drop_requested_keys() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
DsApply (class in eskapade.core_ops.links), 119
DsApply (class in eskapade.core_ops.links.apply), 105
DsObjectDeleter (class in eskapade.core_ops.links), 114
DsObjectDeleter (class in eskapade.core_ops.links.ds_object_deleter), 107
DsToDs (class in eskapade.core_ops.links), 115
DsToDs (class in eskapade.core_ops.links.ds_to_ds), 107
dt_string() (eskapade.analysis.datetime.FreqTimePeriod method), 69

E

Error, 91
eskapade (module), 140
eskapade.analysis (module), 86
eskapade.analysis.correlation (module), 69
eskapade.analysis.datetime (module), 69
eskapade.analysis.histogram (module), 71
eskapade.analysis.histogram_filling (module), 80
eskapade.analysis.links (module), 60
eskapade.analysis.links.apply_func_to_df (module), 47
eskapade.analysis.links.apply_selection_to_df (module), 48
eskapade.analysis.links.basic_generator (module), 49
eskapade.analysis.links.df_concatenator (module), 50
eskapade.analysis.links.df_merger (module), 50
eskapade.analysis.links.histogrammar_filler (module), 51
eskapade.analysis.links.random_sample_splitter (module), 53
eskapade.analysis.links.read_to_df (module), 54
eskapade.analysis.links.record_factorizer (module), 55
eskapade.analysis.links.record_vectorizer (module), 56
eskapade.analysis.links.value_counter (module), 57
eskapade.analysis.links.write_from_df (module), 59
eskapade.analysis.statistics (module), 83
eskapade.core (module), 105
eskapade.core.definitions (module), 86
eskapade.core.element (module), 88
eskapade.core.exceptions (module), 91
eskapade.core.execution (module), 91
eskapade.core.meta (module), 92
eskapade.core.mixin (module), 94
eskapade.core.persistence (module), 96
eskapade.core.process_manager (module), 97
eskapade.core.process_services (module), 100
eskapade.core.run_utils (module), 104
eskapade.core_ops (module), 120
eskapade.core_ops.links (module), 114
eskapade.core_ops.links.apply (module), 105
eskapade.core_ops.links.assert_in_ds (module), 105
eskapade.core_ops.links.break_link (module), 106
eskapade.core_ops.links.ds_object_deleter (module), 106
eskapade.core_ops.links.ds_to_ds (module), 107
eskapade.core_ops.links.event_looper (module), 108
eskapade.core_ops.links.hello_world (module), 109

eskapade.core_ops.links.import_data_store (module), 109
eskapade.core_ops.links.ipython_embed (module), 110
eskapade.core_ops.links.line_printer (module), 110
eskapade.core_ops.links.print_ds (module), 111
eskapade.core_ops.links.repeat_chain (module), 112
eskapade.core_ops.links.skip_chain_if_empty (module), 112
eskapade.core_ops.links.to_ds_dict (module), 113
eskapade.data_quality (module), 125
eskapade.data_quality.dq_helper (module), 124
eskapade.data_quality.links (module), 122
eskapade.data_quality.links.fix_pandas_dataframe (module), 120
eskapade.entry_points (module), 136
eskapade.exceptions (module), 136
eskapade.helpers (module), 137
eskapade.logger (module), 125
eskapade.resources (module), 138
eskapade.utils (module), 139
eskapade.version (module), 139
eskapade.visualization (module), 136
eskapade.visualization.links (module), 130
eskapade.visualization.links.correlation_summary (module), 125
eskapade.visualization.links.df_boxplot (module), 126
eskapade.visualization.links.df_summary (module), 127
eskapade.visualization.vis_utils (module), 133
eskapade_bootstrap() (in module eskapade.entry_points), 136
eskapade_generate_link() (in module eskapade.entry_points), 136
eskapade_generate_macro() (in module eskapade.entry_points), 136
eskapade_generate_notebook() (in module eskapade.entry_points), 136
eskapade_ignite() (in module eskapade.entry_points), 136
eskapade_run() (in module eskapade.core.execution), 91
eskapade_run() (in module eskapade.entry_points), 136
eskapade_trial() (in module eskapade.entry_points), 136
EventLooper (class in eskapade.core_ops.links), 115
EventLooper (class in eskapade.core_ops.links.event_looper), 108
execute() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
execute() (eskapade.analysis.links.apply_func_to_df.ApplyFuncToDf method), 48
execute() (eskapade.analysis.links.apply_selection_to_df.ApplySelectionToDf method), 49
execute() (eskapade.analysis.links.ApplyFuncToDf method), 60
execute() (eskapade.analysis.links.ApplySelectionToDf method), 61
execute() (eskapade.analysis.links.basic_generator.BasicGenerator method), 49
execute() (eskapade.analysis.links.BasicGenerator method), 61
execute() (eskapade.analysis.links.df_concatenator.DfConcatenator method), 50
execute() (eskapade.analysis.links.df_merger.DfMerger method), 51
execute() (eskapade.analysis.links.DfConcatenator method), 62
execute() (eskapade.analysis.links.DfMerger method), 62
execute() (eskapade.analysis.links.random_sample_splitter.RandomSampleS method), 53
execute() (eskapade.analysis.links.RandomSampleSplitter method), 64
execute() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
execute() (eskapade.analysis.links.ReadToDf method), 65
execute() (eskapade.analysis.links.record_factorizer.RecordFactorizer method), 56
execute() (eskapade.analysis.links.record_vectorizer.RecordVectorizer method), 56
execute() (eskapade.analysis.links.RecordFactorizer method), 66
execute() (eskapade.analysis.links.RecordVectorizer method), 66
execute() (eskapade.analysis.links.write_from_df.WriteFromDf method), 59
execute() (eskapade.analysis.links.WriteFromDf method), 68
execute() (eskapade.core.element.Chain method), 89
execute() (eskapade.core.element.Link method), 90
execute() (eskapade.core.meta.Processor method), 92
execute() (eskapade.core.process_manager.ProcessManager method), 98
execute() (eskapade.core_ops.links.apply.DsApply method), 105
execute() (eskapade.core_ops.links.assert_in_ds.AssertInDs method), 106
execute() (eskapade.core_ops.links.AssertInDs method), 114
execute() (eskapade.core_ops.links.Break method), 114
execute() (eskapade.core_ops.links.break_link.Break method), 106
execute() (eskapade.core_ops.links.ds_object_deleter.DsObjectDeleter method), 107
execute() (eskapade.core_ops.links.ds_to_ds.DsToDs method), 108
execute() (eskapade.core_ops.links.DsApply method), 119
execute() (eskapade.core_ops.links.DsObjectDeleter method), 115
execute() (eskapade.core_ops.links.DsToDs method), 115
execute() (eskapade.core_ops.links.event_looper.EventLooper

method), 108
execute() (eskapade.core_ops.links.EventLooper method), 116
execute() (eskapade.core_ops.links.hello_world.HelloWorld method), 109
execute() (eskapade.core_ops.links.HelloWorld method), 116
execute() (eskapade.core_ops.links.import_data_store.ImportDataStore method), 110
execute() (eskapade.core_ops.links.ImportDataStore method), 119
execute() (eskapade.core_ops.links.ipython_embed.IPythonEmbed method), 110
execute() (eskapade.core_ops.links.IPythonEmbed method), 116
execute() (eskapade.core_ops.links.line_printer.LinePrinter method), 111
execute() (eskapade.core_ops.links.LinePrinter method), 117
execute() (eskapade.core_ops.links.print_ds.PrintDs method), 111
execute() (eskapade.core_ops.links.PrintDs method), 117
execute() (eskapade.core_ops.links.repeat_chain.RepeatChain method), 112
execute() (eskapade.core_ops.links.RepeatChain method), 117
execute() (eskapade.core_ops.links.skip_chain_if_empty.SkipChainIfEmpty method), 113
execute() (eskapade.core_ops.links.SkipChainIfEmpty method), 118
execute() (eskapade.core_ops.links.to_ds_dict.ToDsDict method), 114
execute() (eskapade.core_ops.links.ToDsDict method), 119
execute() (eskapade.data_quality.links.fix_pandas_dataframe.FixPandasDataFrame method), 121
execute() (eskapade.data_quality.links.FixPandasDataFrame method), 123
execute() (eskapade.visualization.links.correlation_summary.CorrelationSummary method), 126
execute() (eskapade.visualization.links.CorrelationSummary method), 130
execute() (eskapade.visualization.links.df_boxplot.DfBoxplot method), 127
execute() (eskapade.visualization.links.df_summary.DfSummary method), 128
execute() (eskapade.visualization.links.DfBoxplot method), 131
execute() (eskapade.visualization.links.DfSummary method), 132
execute_macro() (eskapade.core.process_manager.ProcessManager method), 98

F

Failure (eskapade.core.definitions.StatusCode attribute), 87
fill_histogram() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
fill_histogram() (eskapade.analysis.links.histogrammar_filler.Histogrammar method), 52
fill_histogram() (eskapade.analysis.links.HistogrammarFiller method), 63
fill_histogram() (eskapade.analysis.links.value_counter.ValueCounter method), 58
fill_histogram() (eskapade.analysis.links.ValueCounter method), 68
finalize() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
finalize() (eskapade.analysis.links.value_counter.ValueCounter method), 58
finalize() (eskapade.analysis.links.ValueCounter method), 68
finalize() (eskapade.core.element.Chain method), 89
finalize() (eskapade.core.element.Link method), 90
finalize() (eskapade.core.meta.Processor method), 92
finalize() (eskapade.core.process_manager.ProcessManager method), 98
finalize() (eskapade.core_ops.links.event_looper.EventLooper method), 108
finalize() (eskapade.core_ops.links.EventLooper method), 116
finalize() (eskapade.visualization.links.correlation_summary.CorrelationSur method), 126
finalize() (eskapade.visualization.links.CorrelationSummary method), 130
finalize() (eskapade.visualization.links.df_boxplot.DfBoxplot method), 127
finalize() (eskapade.visualization.links.df_summary.DfSummary method), 128
finalize() (eskapade.visualization.links.DfBoxplot method), 131
finalize() (eskapade.visualization.links.DfSummary method), 132
finish() (eskapade.core.process_services.ProcessService method), 103
FixPandasDataFrame (class in eskapade.data_quality.links), 122
FixPandasDataFrame (class in eskapade.data_quality.links.fix_pandas_dataframe), 120
fixture() (in module eskapade.resources), 138
freq (eskapade.analysis.datetime.FreqTimePeriod attribute), 70
FreqTimePeriod (class in eskapade.analysis.datetime), 69

G

get() (eskapade.core.element.Chain method), 89

get() (eskapade.core.process_manager.ProcessManager method), 98
get() (eskapade.core.process_services.ConfigObject method), 101
get_all_columns() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
get_all_columns() (eskapade.visualization.links.df_summary.DfSummary method), 128
get_all_columns() (eskapade.visualization.links.DfSummary method), 132
get_bin_center() (eskapade.analysis.histogram.BinningUtil method), 72
get_bin_count() (eskapade.analysis.histogram.Histogram method), 74
get_bin_edges() (eskapade.analysis.histogram.BinningUtil method), 72
get_bin_edges_range() (eskapade.analysis.histogram.BinningUtil method), 72
get_bin_labels() (eskapade.analysis.histogram.Histogram method), 75
get_bin_range() (eskapade.analysis.histogram.Histogram method), 75
get_bin_vals() (eskapade.analysis.histogram.Histogram method), 75
get_col_props() (eskapade.analysis.statistics.ArrayStats method), 84
get_col_props() (in module eskapade.analysis.statistics), 86
get_config() (eskapade.core.mixin.ConfigMixin method), 95
get_data_type() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 81
get_env_var() (in module eskapade.utils), 139
get_hist_val() (eskapade.analysis.histogram.Histogram method), 75
get_latex_table() (eskapade.analysis.statistics.ArrayStats method), 84
get_latex_table() (eskapade.analysis.statistics.GroupByStats method), 85
get_left_bin_edge() (eskapade.analysis.histogram.BinningUtil method), 72
get_length() (eskapade.visualization.links.df_summary.DfSummary method), 129
get_length() (eskapade.visualization.links.DfSummary method), 132
get_nonone_bin_centers() (eskapade.analysis.histogram.Histogram method), 75
get_nonone_bin_counts() (eskapade.analysis.histogram.Histogram method), 75
get_nonone_bin_edges() (eskapade.analysis.histogram.Histogram method), 75
get_nonone_bin_range() (eskapade.analysis.histogram.Histogram method), 75
get_print_stats() (eskapade.analysis.statistics.ArrayStats method), 84
get_right_bin_edge() (eskapade.analysis.histogram.BinningUtil method), 72
get_sample() (eskapade.visualization.links.df_summary.DfSummary method), 129
get_sample() (eskapade.visualization.links.DfSummary method), 132
get_service_tree() (eskapade.core.process_manager.ProcessManager method), 99
get_services() (eskapade.core.process_manager.ProcessManager method), 99
get_uniform_bin_edges() (eskapade.analysis.histogram.Histogram method), 76
get_values() (eskapade.analysis.histogram.ValueCounts method), 78
get_x_label() (eskapade.analysis.statistics.ArrayStats method), 84
groupbyapply() (eskapade.analysis.links.apply_func_to_df.ApplyFuncToDf method), 48
groupbyapply() (eskapade.analysis.links.ApplyFuncToDf method), 60
GroupByStats (class in eskapade.analysis.statistics), 85

H

HelloWorld (class in eskapade.core_ops.links), 116
HelloWorld (class in eskapade.core_ops.links.hello_world), 109
Histogram (class in eskapade.analysis.histogram), 73
HistogramFillerBase (class in eskapade.analysis.histogram_filling), 80
HistogrammarFiller (class in eskapade.analysis.links), 62
HistogrammarFiller (class in eskapade.analysis.links.histogrammar_filler), 51

I

import_and_update_datastore() (eskapade.core_ops.links.import_data_store.ImportDataStore method), 110

import_and_update_datastore() (eskapade.core_ops.links.ImportDataStore method), 119
import_from_file() (eskapade.core.process_services.ProcessService class method), 103
import_services() (eskapade.core.process_manager.ProcessManager method), 99
ImportDataStore (class in eskapade.core_ops.links), 119
ImportDataStore (class in eskapade.core_ops.links.import_data_store), 109
initialize() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 82
initialize() (eskapade.analysis.links.apply_func_to_df.ApplyFuncToDf method), 48
initialize() (eskapade.analysis.links.apply_selection_to_df.ApplySelectionToDf method), 49
initialize() (eskapade.analysis.links.ApplyFuncToDf method), 60
initialize() (eskapade.analysis.links.ApplySelectionToDf method), 61
initialize() (eskapade.analysis.links.basic_generator.BasicGenerator method), 49
initialize() (eskapade.analysis.links.BasicGenerator method), 61
initialize() (eskapade.analysis.links.df_concatenator.DfConcatenator method), 50
initialize() (eskapade.analysis.links.df_merger.DfMerger method), 51
initialize() (eskapade.analysis.links.DfConcatenator method), 62
initialize() (eskapade.analysis.links.DfMerger method), 62
initialize() (eskapade.analysis.links.random_sample_splitter.RandomSampleSplitter method), 53
initialize() (eskapade.analysis.links.RandomSampleSplitter method), 64
initialize() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
initialize() (eskapade.analysis.links.ReadToDf method), 65
initialize() (eskapade.analysis.links.record_factorizer.RecordFactorizer method), 56
initialize() (eskapade.analysis.links.record_vectorizer.RecordVectorizer method), 56
initialize() (eskapade.analysis.links.RecordFactorizer method), 66
initialize() (eskapade.analysis.links.RecordVectorizer method), 66
initialize() (eskapade.analysis.links.value_counter.ValueCounter method), 58
initialize() (eskapade.analysis.links.ValueCounter method), 68
initialize() (eskapade.analysis.links.write_from_df.WriteFromDf method), 59
initialize() (eskapade.analysis.links.WriteFromDf method), 68
initialize() (eskapade.core.element.Chain method), 89
initialize() (eskapade.core.element.Link method), 90
initialize() (eskapade.core.meta.Processor method), 92
initialize() (eskapade.core.process_manager.ProcessManager method), 99
initialize() (eskapade.core_ops.links.ds_object_deleter.DsObjectDeleter method), 107
initialize() (eskapade.core_ops.links.ds_to_ds.DsToDs method), 108
initialize() (eskapade.core_ops.links.DsObjectDeleter method), 115
initialize() (eskapade.core_ops.links.DsToDs method), 115
initialize() (eskapade.core_ops.links.event_looper.EventLooper method), 108
initialize() (eskapade.core_ops.links.EventLooper method), 116
initialize() (eskapade.core_ops.links.import_data_store.ImportDataStore method), 110
initialize() (eskapade.core_ops.links.ImportDataStore method), 119
initialize() (eskapade.core_ops.links.line_printer.LinePrinter method), 111
initialize() (eskapade.core_ops.links.LinePrinter method), 117
initialize() (eskapade.core_ops.links.repeat_chain.RepeatChain method), 112
initialize() (eskapade.core_ops.links.RepeatChain method), 117
initialize() (eskapade.core_ops.links.skip_chain_if_empty.SkipChainIfEmpt method), 113
initialize() (eskapade.core_ops.links.SkipChainIfEmpty method), 118
initialize() (eskapade.core_ops.links.to_ds_dict.ToDsDict method), 114
initialize() (eskapade.core_ops.links.ToDsDict method), 119
initialize() (eskapade.data_quality.links.fix_pandas_dataframe.FixPandasDa method), 122
initialize() (eskapade.data_quality.links.FixPandasDataFrame method), 124
initialize() (eskapade.visualization.links.correlation_summary.CorrelationSu method), 126
initialize() (eskapade.visualization.links.CorrelationSummary method), 130
initialize() (eskapade.visualization.links.df_boxplot.DfBoxplot method), 127
initialize() (eskapade.visualization.links.df_summary.DfSummary method), 129

initialize() (eskapade.visualization.links.DfBoxplot method), 131
initialize() (eskapade.visualization.links.DfSummary method), 132
io_base_dirs() (eskapade.core.process_services.ConfigObject method), 101
io_conf() (eskapade.core.process_services.ConfigObject method), 102
io_dir() (in module eskapade.core.persistence), 96
io_path() (in module eskapade.core.persistence), 96
IPythonEmbed (class in eskapade.core_ops.links), 116
IPythonEmbed (class in eskapade.core_ops.links.ipython_embed), 110
is_finished() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
is_finished() (eskapade.analysis.links.ReadToDf method), 65

K

key (eskapade.analysis.histogram.ValueCounts attribute), 79

L

latest_data_length() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
latest_data_length() (eskapade.analysis.links.ReadToDf method), 65
lib() (in module eskapade.resources), 138
LinePrinter (class in eskapade.core_ops.links), 116
LinePrinter (class in eskapade.core_ops.links.line_printer), 111
Link (class in eskapade.core.element), 89
load() (eskapade.core.element.Link method), 90
logger (eskapade.analysis.datetime.TimePeriod attribute), 70
logger (eskapade.analysis.histogram.Histogram attribute), 76
logger (eskapade.analysis.statistics.ArrayStats attribute), 84
logger (eskapade.core.meta.Processor attribute), 92
logger (eskapade.core.process_services.ProcessService attribute), 103

M

make_histogram() (eskapade.analysis.statistics.ArrayStats method), 85
MissingPackageError, 136

N

n_bins (eskapade.analysis.histogram.Histogram attribute), 77
n_chains (eskapade.core.process_manager.ProcessManager attribute), 99
n_dim (eskapade.analysis.histogram.Histogram attribute), 77
n_links (eskapade.core.element.Chain attribute), 89
name (eskapade.core.meta.Processor attribute), 93
nononecounts (eskapade.analysis.histogram.ValueCounts attribute), 79
num_bins (eskapade.analysis.histogram.Histogram attribute), 77
num_bins (eskapade.analysis.histogram.ValueCounts attribute), 79
num_nonone_bins (eskapade.analysis.histogram.ValueCounts attribute), 79

O

obj_repr() (in module eskapade.helpers), 137
offset (eskapade.analysis.datetime.UniformTsTimePeriod attribute), 71
only_bool() (in module eskapade.analysis.histogram_filling), 82
only_float() (in module eskapade.analysis.histogram_filling), 82
only_int() (in module eskapade.analysis.histogram_filling), 82
only_str() (in module eskapade.analysis.histogram_filling), 82

P

pandas_writer() (in module eskapade.analysis.links.write_from_df), 60
pandasReader() (in module eskapade.analysis.links.read_to_df), 55
parent (eskapade.core.meta.Processor attribute), 93
parse_date_time() (eskapade.analysis.datetime.TimePeriod class method), 71
parse_time_period() (eskapade.analysis.datetime.TimePeriod class method), 71
period (eskapade.analysis.datetime.UniformTsTimePeriod attribute), 71
period_index() (eskapade.analysis.datetime.FreqTimePeriod method), 70
period_index() (eskapade.analysis.datetime.TimePeriod method), 71
period_index() (eskapade.analysis.datetime.UniformTsTimePeriod method), 71
persist (eskapade.core.process_services.ProcessServiceMeta attribute), 104
persist_in_file() (eskapade.core.process_services.ProcessService method), 104

persist_services() (eskapade.core.process_manager.ProcessManager method), 99
plot_2d_histogram() (in module eskapade.visualization.vis_utils), 134
plot_correlation_matrix() (in module eskapade.visualization.vis_utils), 134
plot_histogram() (in module eskapade.visualization.vis_utils), 135
pop() (eskapade.core.meta.ProcessorSequence method), 94
Print() (eskapade.core.process_services.ConfigObject method), 101
Print() (eskapade.core.process_services.DataStore method), 102
print_chains() (eskapade.core.process_manager.ProcessManager method), 99
print_services() (eskapade.core.process_manager.ProcessManager method), 99
PrintDs (class in eskapade.core_ops.links), 117
PrintDs (class in eskapade.core_ops.links.print_ds), 111
process_1d_histogram() (eskapade.visualization.links.df_summary.DfSummary method), 129
process_1d_histogram() (eskapade.visualization.links.DfSummary method), 132
process_2d_histogram() (eskapade.visualization.links.df_summary.DfSummary method), 129
process_2d_histogram() (eskapade.visualization.links.DfSummary method), 132
process_and_store() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 82
process_and_store() (eskapade.analysis.links.value_counter.ValueCounter method), 59
process_and_store() (eskapade.analysis.links.ValueCounter method), 68
process_columns() (eskapade.analysis.histogram_filling.HistogramFillerBase method), 82
process_columns() (eskapade.analysis.links.value_counter.ValueCounter method), 59
process_columns() (eskapade.analysis.links.ValueCounter method), 68
process_counts() (eskapade.analysis.histogram.ValueCounts method), 79
process_nan_histogram() (eskapade.visualization.links.df_summary.DfSummary method), 129
process_nan_histogram() (eskapade.visualization.links.DfSummary method), 133
process_sample() (eskapade.visualization.links.df_summary.DfSummary method), 129
process_sample() (eskapade.visualization.links.DfSummary method), 133
process_series() (eskapade.visualization.links.df_summary.DfSummary method), 129
process_series() (eskapade.visualization.links.DfSummary method), 133
process_transform_funcs() (in module eskapade.helpers), 137
ProcessManager (class in eskapade.core.process_manager), 97
Processor (class in eskapade.core.meta), 92
ProcessorSequence (class in eskapade.core.meta), 93
ProcessService (class in eskapade.core.process_services), 102
ProcessServiceMeta (class in eskapade.core.process_services), 104

R

RandomSampleSplitter (class in eskapade.analysis.links), 64
RandomSampleSplitter (class in eskapade.analysis.links.random_sample_splitter), 53
RandomSeeds (class in eskapade.core.definitions), 86
ReadToDf (class in eskapade.analysis.links), 64
ReadToDf (class in eskapade.analysis.links.read_to_df), 54
record_file_number() (in module eskapade.core.persistence), 97
record_vectorizer() (in module eskapade.analysis.links.record_vectorizer), 57
RecordFactorizer (class in eskapade.analysis.links), 65
RecordFactorizer (class in eskapade.analysis.links.record_factorizer), 55
RecordVectorizer (class in eskapade.analysis.links), 66
RecordVectorizer (class in eskapade.analysis.links.record_vectorizer), 56
Recoverable (eskapade.core.definitions.StatusCode attribute), 87
remove_all_services() (eskapade.core.process_manager.ProcessManager method), 99
remove_keys_of_inconsistent_type() (eskapade.analysis.histogram.Histogram method),

77
remove_keys_of_inconsistent_type() (eskapade.analysis.histogram.ValueCounts method), 79
remove_service() (eskapade.core.process_manager.ProcessManager method), 99
RepeatChain (class in eskapade.core_ops.links), 117
RepeatChain (class in eskapade.core_ops.links.repeat_chain), 112
RepeatChain (eskapade.core.definitions.StatusCode attribute), 87
repl_whites() (in module eskapade.core.persistence), 97
reset() (eskapade.core.process_manager.ProcessManager method), 100
reset_config() (eskapade.core.mixin.ConfigMixin method), 95
reset_eskapade() (in module eskapade.core.execution), 92
rho_from_chi2() (in module eskapade.analysis.correlation), 69
run() (eskapade.core.element.Link method), 90
run() (eskapade.core.process_manager.ProcessManager method), 100

S

service() (eskapade.core.process_manager.ProcessManager method), 100
set_begin_end_chain_opt() (in module eskapade.core.definitions), 87
set_chunk_size() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
set_chunk_size() (eskapade.analysis.links.ReadToDf method), 65
set_custom_user_vars() (in module eskapade.core.definitions), 87
set_log_level_opt() (in module eskapade.core.definitions), 87
set_matplotlib_backend() (in module eskapade.utils), 139
set_opt_var() (in module eskapade.core.definitions), 87
set_seeds() (in module eskapade.core.definitions), 87
set_single_chain_opt() (in module eskapade.core.definitions), 87
set_user_opts() (eskapade.core.process_services.ConfigObject method), 102
simulate() (eskapade.analysis.histogram.Histogram method), 77
Singleton (class in eskapade.core.meta), 94
skey (eskapade.analysis.histogram.ValueCounts attribute), 79
SkipChain (eskapade.core.definitions.StatusCode attribute), 87
SkipChainIfEmpty (class in eskapade.core_ops.links), 117
SkipChainIfEmpty (class in eskapade.core_ops.links.skip_chain_if_empty), 112
start_timer() (eskapade.core.mixin.TimerMixin method), 96
StatusCode (class in eskapade.core.definitions), 87
stop_timer() (eskapade.core.mixin.TimerMixin method), 96
store() (eskapade.core.element.Link method), 91
Success (eskapade.core.definitions.StatusCode attribute), 87
sum_counts (eskapade.analysis.histogram.ValueCounts attribute), 79
sum_data_length() (eskapade.analysis.links.read_to_df.ReadToDf method), 54
sum_data_length() (eskapade.analysis.links.ReadToDf method), 65
sum_nonone_counts (eskapade.analysis.histogram.ValueCounts attribute), 79
summary() (eskapade.core.element.Link method), 91
summary() (eskapade.core.process_manager.ProcessManager method), 100
surface() (eskapade.analysis.histogram.Histogram method), 77

T

template() (in module eskapade.resources), 138
TimePeriod (class in eskapade.analysis.datetime), 70
TimerMixin (class in eskapade.core.mixin), 95
to_date_time() (in module eskapade.data_quality.dq_helper), 125
to_float() (in module eskapade.data_quality.dq_helper), 125
to_int() (in module eskapade.data_quality.dq_helper), 125
to_normalized() (eskapade.analysis.histogram.Histogram method), 77
to_ns() (in module eskapade.analysis.histogram_filling), 82
to_str() (in module eskapade.analysis.histogram_filling), 82
to_str() (in module eskapade.data_quality.dq_helper), 125
ToDsDict (class in eskapade.core_ops.links), 118
ToDsDict (class in eskapade.core_ops.links.to_ds_dict), 113
total_time() (eskapade.core.mixin.TimerMixin method), 96
truncated_bin_edges() (eskapade.analysis.histogram.BinningUtil method), 73
tutorial() (in module eskapade.resources), 139

U
Undefined (eskapade.core.definitions.StatusCode attribute), 87
UniformTsTimePeriod (class in eskapade.analysis.datetime), 71
UnknownSetting, 91

V
value_to_bin_center() (in module eskapade.analysis.histogram_filling), 83
value_to_bin_index() (in module eskapade.analysis.histogram_filling), 83
value_to_bin_label() (eskapade.analysis.histogram.BinningUtil method), 73
ValueCounter (class in eskapade.analysis.links), 66
ValueCounter (class in eskapade.analysis.links.value_counter), 57
ValueCounts (class in eskapade.analysis.histogram), 78
variable (eskapade.analysis.histogram.Histogram attribute), 77

W
weighted_quantile() (in module eskapade.analysis.statistics), 86
WriteFromDf (class in eskapade.analysis.links), 68
WriteFromDf (class in eskapade.analysis.links.write_from_df), 59