Actio Python Utils Documentation

Release 0.1.0

Brett Copeland

Mar 05, 2024

CONTENTS:

CHAPTER

ONE

ACTIO PYTHON UTILS

Various Python utility functions mostly about `argparse`, `logging`, `psycopg2`, and `pyspark`. Can be installed via `pip` e.g. `pip install git+ssh://git@github.com/ActioBio/actio_python_utils.git`.

1.1 Features

- TODO

CHAPTER TWO

INSTALLATION

2.1 Stable release

To install Actio Python Utils, run this command in your terminal:

```
$ pip install actio_python_utils
```

This is the preferred method to install Actio Python Utils, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Actio Python Utils can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/brcopeland/actio_python_utils
```

Or download the tarball:

```
$ curl -OJL https://github.com/brcopeland/actio_python_utils/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER THREE

USAGE

To use Actio Python Utils in a project:

```
import actio_python_utils
```

To connect to the DB use `actio_python_utils.database_functions.DBConnection` class:

`actio_python_utils.database_functions.`**`DBConnection`**(*service: str* | *None = None*, *db_args: ~collections.abc.Mapping[str, ~typing.Any]* | *None = None*, *log_name: str = 'sql_debug'*, *commit: bool = False*, *cursor_factory: ~psycopg2.extensions.cursor = <class 'actio_python_utils.database_functions.LoggingCursor'>*) → None

Creates a psycopg2 database connection with the specified parameters and acts as a context manager.

Parameters

- **`service`** (*`str or None`*) – The PostgreSQL service to connect to, defaults to None
- **`db_args`** (*`Mapping or None`*) – A mapping of database connection arguments, defaults to None
- **`log_name`** (*`str`*) – The name to give the logger, defaults to cfg["logging"]["names"]["db"]
- **`commit`** (*`bool`*) – Commit the transaction upon closing the connection if no errors were encountered, defaults to False
- **`psycopg2.cursor_factory`** – The class of cursor to use for the connection by default (used for service/$DB_CONNECTION_STRING, defaults to LoggingCursor

Raises

`ValueError` – If service and db_args are specified

CHAPTER FOUR

API

actio_python_utils	Top-level package for Actio Python Utils.

4.1 actio_python_utils

Top-level package for Actio Python Utils.

Modules

actio_python_utils.actio_python_utils	Main module.
actio_python_utils.argparse_functions	Code for argparse-related functionality.
actio_python_utils.cli	Console script for actio_python_utils.
actio_python_utils.database_functions	Database-related functionality.
actio_python_utils.logging_functions	Logging-related functionality.
actio_python_utils.spark_functions	Spark-related functionality.
actio_python_utils.utils	

4.1.1 actio_python_utils.actio_python_utils

Main module.

4.1.2 actio_python_utils.argparse_functions

Code for argparse-related functionality.

Functions

`dir_exists`(dirn)	Returns the real path to directory dirn if it exists
`file_exists`(fn)	Returns the real path to file fn if it exists
`key_value_pair`(arg[, sep])	Splits a string once on sep and returns the result
`str_from_file`(fn)	Returns the text from a file name

Classes

`CustomFormatter`(prog[, indent_increment, ...])	argparse.HelpFormatter that displays argument defaults and doesn't change formatting of the description
`EnhancedArgumentParser`(*args, description, ...)	Customized ArgumentParser that sets description automatically, uses both ArgumentDefaultsHelpFormatter and RawTextHelpFormatter formatters, optionally sets up logging, database, and Spark connections.
`ZFileType`([mode, bufsize, encoding, errors])	

4.1.3 actio_python_utils.cli

Console script for actio_python_utils.

Functions

`main`()	Console script for actio_python_utils.

4.1.4 actio_python_utils.database_functions

Database-related functionality.

Functions

connect_to_db([service, db_args])	Return a connection to the specified PostgreSQL database
get_db_args([service, db_args, logger, ...])	Returns a dict of arguments to log in with psycopg2. Resolution order for connecting to a database is as follows: 1a. service, referring to a PostgreSQL service name, normally defined in ~/.pg_service.conf 1b. db_args, arbitrary dictionary of arguments 2. Environment variable DB_CONNECTION_STRING with format postgres://your_user:your_password@your_host:your_port/your_database 3. Environment variable PGSERVICE, referring to a PostgreSQL service name 4. cfg["db"], which should resolve to a dictionary of arguments.
get_pg_config([service, service_fn, pgpass_fn])	Locates the PostgreSQL login credentials given a service name Uses service = $PGSERVICE or cfg["db"]["service"] if not specified.
replace_dict_values_with_global_definition(n	Replaces values in the mapping with those in global_mapping when the value is a str and it is a key in the global mapping
savepoint(cur[, savepoint])	Creates a context manager to create a savepoint upon entering, rollback if an error occurs, and release upon exiting
savepoint_wrapper(method)	Wraps a psycopg2 cursor's method to use a savepoint
split_schema_from_table(table[, default_schema])	Split a possibly schema qualified table name into its schema and table names

Classes

DBConnection(service, db_args, ...)	Creates a psycopg2 database connection with the specified parameters and acts as a context manager.
LoggingCursor(*args[, log_level, log_name])	Wraps a DictCursor such that each copy_expert and execute statement is logged
SavepointCursor(*args[, log_level, log_name])	Wraps LoggingCursor methods to use a savepoint context manager

4.1.5 actio_python_utils.logging_functions

Logging-related functionality.

Functions

`log`(level)	Used as a context manager to log at the specified level temporarily and return to the previous level after exiting
`setup_logging`([logging_level, name, stream, ...])	Set up the logger given by name, attach a stream handler, set the format, and log levels as specified.

Classes

`LazyLogger`(name[, level])	Wraps a Logger to accept either a message or a function to evaluate to produce the desired message.

4.1.6 actio_python_utils.spark_functions

Spark-related functionality.

Functions

`convert_chromosome`(self, current_column_name)	Return a PySpark dataframe with current_column_name (containing human chromosomes) with a new column, new_column_name (defaulting to overwriting the original), with the chromosome cast as an integer.
`convert_dicts_to_dataframe`(self[, ...])	Converts either a list of dicts (dict_list) or a function that returns an iterator of dicts (iter_func) to a PySpark dataframe
`count_columns_with_string`(self[, string])	Return a PySpark dataframe with the number of times a given string occurs in each string column in a dataframe
`count_distinct_values`(self[, ...])	Return a new PySpark dataframe with the number of distinct values in each column.
`count_nulls`(self)	Return a PySpark dataframe with the number of null values in each column of a dataframe
`load_dataframe`(self, path[, format, ...])	Load and return the specified data source using PySpark
`load_db_to_dataframe`(self[, pgpass_record, ...])	Return a PySpark dataframe from either a relation or query
`load_excel_to_dataframe`(self, xl_fn[, ...])	Load and return the specified Excel spreadsheet with PySpark
`load_xml_to_dataframe`(self, xml_fn, row_tag)	Load and return the specified XML file with PySpark
`serialize_array_field`(self, column, ...[, ...])	Serializes an ArrayType field for output.
`serialize_bool_field`(self, column, new_column)	Serializes a BooleanType field for output.
`serialize_field`(self, column[, new_column, ...])	Operates on a PySpark dataframe and converts any field of either atoms or structs, or any array of either of those (but not nested) to the properly formatted string for postgresql TEXT loading format and assigns it the column name new_column.
`serialize_string_field`(self, column, new_column)	Serializes a StringType field for output.
`serialize_struct_field`(self, column, ...[, ...])	Serializes a StructType field for output.
`setup_spark`([cores, memory, use_db, ...])	Configures and creates a PySpark session according to the supplied arguments
`split_dataframe_to_csv_by_column_value`(self, ...)	Split a dataframe with PySpark to a set of gzipped CSVs, e.g. if a dataframe has data: col1,col2,col3 1,1,1 1,2,3 2,1,1.

4.1.7 actio_python_utils.utils

Functions

`cast_chromosome_to_int`(chromosome)	Cast a chromosome string, optionally prefixed with chr, to an integer.
`check_valid_output_directory`(output_directory)	Check if the given directory is valid for outputting.
`coalesce`(*args)	Return the first argument that is not None
`debug`(func)	Wraps a function to output the function signature, run the function, output the return value, and return the return value.
`extract_excel_sheet`(fn, output_fn[, sheet, ...])	Extract a sheet from an Excel spreadsheet.
`flatten_cfg`(key[, d, sep])	Flatten a nested value in a dict
`get_csv_fields`(fn[, sep, sanitize, ...])	Get the column names from the first line of the specified file and optionally replaces an arbitrary sequence of strings to others
`open_bz2`(filename[, mode, buff, external])	Return a file handle to filename using pbzip2, bzip2, or b2 module
`open_gz`(filename[, mode, buff, external])	Return a file handle to filename using pigz, gzip, or gzip module
`open_pipe`(command[, mode, buff])	Runs a subprocess.Popen and either retains input or output
`open_xz`(filename[, mode, buff, external])	Return a file handle to filename using either xz or lzma module
`rename_dict_keys`(original_dict, renames)	Returns a new dict that's a copy of the supplied dict but with an arbitrary number of keys renamed
`sync_to_s3`(dir_name, s3_path)	Syncs a directory to specific S3 bucket/path.
`timer`(func)	Wraps a function to output the running time of function calls.
`which`(program)	**param str program** The program to find
`zopen`(filename[, mode, buff, external])	Open pipe, zipped, or unzipped file automagically

Classes

`CustomCSVDialect`()	
`DictToFunc`(choices)	Class that after initializing with a dict can be called to map keys to values, e.g. `` ` D = DictToFunc({"a": 42, "b": "apple"}) D("a") 42 D("b") "apple" D("c") KeyError("c") ` ``.
`NumericValue`(min_value, max_value, left_op, ...)	Creates a class that can be used as a function to verify that a passed argument is a numeric value of the correct type and in the expected range

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/ActioBio/actio_python_utils/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Actio Python Utils could always use more documentation, whether as part of the official Actio Python Utils docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ActioBio/actio_python_utils/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *actio_python_utils* for local development.

1. Fork the *actio_python_utils* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/actio_python_utils.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv actio_python_utils
   $ cd actio_python_utils/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ git checkout -b name-of-your-bugfix-or-feature
   ```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 actio_python_utils tests
   $ python setup.py test or pytest
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

   ```
   $ git add .
   $ git commit -m "Your detailed description of your changes."
   $ git push origin name-of-your-bugfix-or-feature
   ```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/ActioBio/actio_python_utils/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_actio_python_utils
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Brett Copeland <brcopeland@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2023-06-21)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX