ICCAS Python Helper Documentation

Release 0.2.0

Gianluca Gippetto

Dec 09, 2020

CONTENTS:

CHAPTER

ONE

ICCAS PYTHON HELPER

This repository contains:

- a helper package to get the ICCAS dataset (Italian Coronavirus Cases by Age group and Sex), processing and visualizing it;
- some Jupyter notebooks that you can run on Binder clicking on the badge "launch notebooks" above.

1.1 The package

The package includes several submodules:

Module	Description
`loading`	Obtain, cache and load the dataset(s)
`processing`	Data (pre)processing. Fix data inconsistencies, resample data with interpolation.
`queries`	Select subsets of data, aggregate or extract useful values.
`charts`	Draw charts and animations (in Italian or English).

To install the package:

```
pip install iccas
```

If you want to use the CLI, you need extra dependencies:

```
pip install iccas[cli]
```

and you also need to install ffmpeg.

- Free software: MIT license
- Documentation: https://iccas-python.readthedocs.io/en/latest/readme.html

1.2 Notebooks

Notebooks text is written in Italian but charts are available in English as well. You just need to run in the first cell:

```
ic.set_locale('en')
```

To run notebooks locally, you need to install jupyter , for example with:

```
pip install jupyterlab
```

Then:

```
pip install -r binder/requirements.txt
./binder/postBuild
```

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER

TWO

ICCAS

2.1 iccas package

2.1.1 Subpackages

iccas.charts package

Submodules

iccas.charts.area module

iccas.charts.area.**area_chart**(*counts_over_time*, *, *strings*, *normalize=False*, *ax=None*, *cmap='inferno'*, *alpha=0.8*, ***kwargs*)

A base for stacked area charts (tuned for iccas charts).

Parameters

- **counts_over_time** (DataFrame) – a DataFrame with a DateTimeIndex and a column for each age group (e.g. ic.get().cases)
- **strings** (*Translation*) –
- **normalize** (bool) –
- **ax** (Optional[Axes]) –
- **cmap** –
- **alpha** (float) –
- ****kwargs** –

Returns:

Return type Axes

iccas.charts.area.**double_area_chart_of_cumulative_counts**(*data*, *variable='cases'*, *, *age_group_size=20*, *period=None*, *strings*, ***figure_args*)

Not a very interesting chart.

Parameters

- **data** (DataFrame) – DataFrame having 'cases' and/or 'deaths' as first-level columns (e.g. ic.get())

- **variable** (str) –
- **age_group_size** (int) –
- **period** (Optional[Tuple[Union[str, Timestamp], Union[str, Timestamp]]]) –
- **strings** (*Translation*) –
- **lang** –

Returns:

Return type Figure

iccas.charts.area.**double_area_chart_of_running_averages**(*data*, *variable='cases'*, ***, *strings*, *window=14*, *age_group_size=20*, *period=None*, ***figure_args*)

iccas.charts.bar module

iccas.charts.bar.**add_labels_to_bars**(*rects*, *fmt='{:.1f}'*, *ax=None*, *y_offset=7*, ***kwargs*)

Attach a text label above each bar in rects, displaying its height. Returns a tuple with the list of labels and a function to update the labels (useful for animations)

Return type Tuple[List[Annotation], Callable[[], None]]

iccas.charts.bar.**age_dist_bar_chart**(*counts*, *date*, *variable='cases'*, ***, *normalize=True*, *age_group_size=10*, *window=14*, *population_distribution=None*, *ax=None*, *lang=None*)

iccas.charts.bar.**average_by_period_bar_chart**(*counts*, *variable*, ***, *strings*, *freq=7*, *normalize=False*, *age_group_size=20*, *stacked=True*, *ylim=None*, *ax=None*, *figsize=12, 7*)

Return type Axes

iccas.charts.common module

iccas.charts.common.**check_age_group_size**(*size*)

iccas.charts.common.**check_variable**(*variable*)

iccas.charts.common.**legend**(*ax=None*, ***kwargs*)

Legend positioned at the right of the axes.

iccas.charts.common.**resample_if_needed**(*data*, *freq='1D'*, ***kwargs*)

Module contents

iccas.charts.**age_dist_bar_chart**(*counts*, *date*, *variable='cases'*, ***, *normalize=True*, *age_group_size=10*, *window=14*, *population_distribution=None*, *ax=None*, *lang=None*)

iccas.charts.**area_chart**(*counts_over_time*, ***, *strings*, *normalize=False*, *ax=None*, *cmap='inferno'*, *alpha=0.8*, ***kwargs*)

A base for stacked area charts (tuned for iccas charts).

Parameters

- **counts_over_time** (DataFrame) – a DataFrame with a DateTimeIndex and a column for each age group (e.g. `ic.get().cases`)
- **strings** (*Translation*) –
- **normalize** (bool) –
- **ax** (Optional[Axes]) –
- **cmap** –
- **alpha** (float) –
- ****kwargs** –

Returns:

Return type Axes

iccas.charts.**average_by_period_bar_chart**(*counts*, *variable*, ***, *strings*, *freq=7*, *normalize=False*, *age_group_size=20*, *stacked=True*, *ylim=None*, *ax=None*, *figsize=12, 7*)

Return type Axes

iccas.charts.**double_area_chart_of_cumulative_counts**(*data*, *variable='cases'*, ***, *age_group_size=20*, *period=None*, *strings*, ***figure_args*)

Not a very interesting chart.

Parameters

- **data** (DataFrame) – DataFrame having 'cases' and/or 'deaths' as first-level columns (e.g. `ic.get()`)
- **variable** (str) –
- **age_group_size** (int) –
- **period** (Optional[Tuple[Union[str, Timestamp], Union[str, Timestamp]]]) –
- **strings** (*Translation*) –
- **lang** –

Returns:

Return type Figure

iccas.charts.**double_area_chart_of_running_averages**(*data*, *variable='cases'*, ***, *strings*, *window=14*, *age_group_size=20*, *period=None*, ***figure_args*)

iccas.cli package

Submodules

iccas.cli.main module

Module contents

iccas.i18n package

Submodules

iccas.i18n.lib module

A simple and crappy i18n library written "for fun" and because neither gettext and python-i18n were optimal for my use case which requires reloading of translations when a change is made without restarting the interpreter/kernel and possibly without reloading the module.

class iccas.i18n.lib.**NullTranslation**
Bases: *iccas.i18n.lib.Translation*

get(*str_id*, *count*, *few_max=5*, *varname='count'*)

Return type str

class iccas.i18n.lib.**Translation**(*lang*, *strings*)
Bases: object

get(*str_id*, *count*, *few_max=5*, *varname='count'*)

Return type str

class iccas.i18n.lib.**TranslationFile**(*path*, *content*, *last_modified*)
Bases: object

content: dict

last_modified: int

static load(*path*)

Return type *TranslationFile*

path: pathlib.Path

static read(*path*)

reload_if_modified()

Return type bool

class iccas.i18n.lib.**TranslationsManager**(**paths*)
Bases: object

injector(*scoped_translations=None*)
Decorator that adds a lang argument to the signature and injects a strings argument of type *Translation*. If the translation files are modified, they are automatically reloaded.

You can also pass extra translations as dict or YAML string to the wrapped function. Nonetheless, any modification will require to reload the module to take effect.

Parameters **scoped_translations** (Union[None, str, Mapping[str, Mapping[str, Union[str, Mapping[str, str]]]]]) – yaml string or dictionary

Module contents

iccas.i18n.**language**(*lang*)

iccas.i18n.**set_language**(*lang*)
Sets the language. Supported languages: Italian (“it”) and English (“en”)

iccas.i18n.**set_locale**(*lang*)
Apart from setting the internal language of the package, also sets the locale accordingly so that pandas/matplotlib displays translated dates

2.1.2 Submodules

2.1.3 iccas.checks module

Sanity checks.

iccas.checks.**is_non_decreasing**(*df*)

iccas.checks.**totals_not_less_than_sum_of_sexes**(*data*, *variable*)

2.1.4 iccas.loading module

iccas.loading.**get**(*cache_dir=PosixPath('/home/docs/.iccas')*)
Returns the latest version of the ICCAS dataset in a pandas.DataFrame (as it’s returned by *load()*).

This function uses RemoteFolderCache.get(), which caches.

Raises

- **request.exceptions.ConnectionError** – if the server is unreachable
- **and no dataset is available in cache_dir** –

Return type DataFrame

iccas.loading.**get_by_date**(*date*, *keep_date=False*, *cache_dir=PosixPath('/home/docs/.iccas')*)

Return type Tuple[DataFrame, Timestamp]

iccas.loading.**get_population_by_age**(*cache_dir=PosixPath('/home/docs/.iccas')*)
Returns a DataFrame with “age” as index and two columns: “value” (absolute counts) and “percentage” (<=1.0)

Return type DataFrame

iccas.loading.**get_population_by_age_group**(*cache_dir=PosixPath('/home/docs/.iccas')*)
Returns a DataFrame with “age_group” as index and two columns: “value” (absolute counts) and “percentage” (<=1.0)

Return type DataFrame

iccas.loading.**get_url**(*date=None*, *fmt='csv'*)
Returns the url of a dataset in a given format. If *date* is None, returns the URL of the full dataset.

Return type str

iccas.loading.**load**(*path*)

Return type DataFrame

iccas.loading.**load_single_date**(*path*, *keep_date=False*)

Loads a dataset containing data for a single date.

By default (*keep_date=False*), the *date* column is dropped and the datetime is stored in the *attrs* of the DataFrame. If instead *keep_date=True*, the returned dataset has a MultiIndex *(date, age_group)*.

Parameters

- **path** (Union[str, Path]) –
- **keep_date** (bool) – whether to drop the date column (containing a single datetime value)

Return type DataFrame

2.1.5 iccas.processing module

iccas.processing.**fix_monotonicity**(*data*, *method='pchip'*, ***interpolation*)

Replaces tracts of all cases and deaths time series that break the non-decreasing trend of the series with interpolated data. This function also ensures that the following conditions are still satisfied even after the "correction":

```
male_cases + female_cases <= cases
male_deaths + female_deaths <= deaths
```

Non-integer columns, if present, are ignored and returned as they are in the output DataFrame.

Parameters

- **data** (DataFrame) – a DataFrame containing all integer columns about cases and deaths
- **method** – interpolation method

Returns a DataFrame with all integer time series (columns) modified so that they are non-decreasing time series

iccas.processing.**nullify_local_bumps**(*df*)

iccas.processing.**nullify_series_local_bumps**(*series*)

Set to NaN all elements s[i] such that s[i] > s[i+k]

iccas.processing.**reindex_by_interpolating**(*data*, *new_index*, *preserve_ints=True*, *method='pchip'*, ***interpolation*)

Reindexes *data* and fills new values by interpolation (PCHIP, by default).

This function was motivated by the fact that `pandas.DataFrame.resample()` followed by `pandas.DataFrame.resample()` doesn't take into account misaligned datetimes.

Parameters

- **data** (*~PandasObj*) – a DataFrame or Series with a datetime index
- **new_index** (DatetimeIndex) –
- **preserve_ints** (bool) – after interpolation, columns containing integers in the original dataframe are rounded and converted back to int
- **method** – interpolation method (see `pandas.DataFrame.interpolate()`)
- ****interpolation** – other interpolation keyword argument different from *method* passed to `pandas.DataFrame.interpolate()`

Return type ~PandasObj

Returns a new Dataframe/Series

See also:

reindex_by_interpolating()

iccas.processing.**resample**(*data*, *freq='1D'*, *hour=18*, *preserve_ints=True*, *method='pchip'*, ***interpolation*)

Resamples *data* and fills missing values by interpolation.

The resulting index is a *pandas.DatetimeIndex* whose elements are spaced accordingly to *freq* and having the time set to *{hour}:00*.

In the case of "day frequencies" ('{num}D'), the index always includes the latest date (*data.index[-1]*): the new index is a datetime range built going backwards from the latest date.

This function was motivated by the fact that `pandas.DataFrame.resample()` followed by `pandas.DataFrame.resample()` doesn't take into account misaligned datetimes. If you want to back-fill or forward-fill, just use `DataFrame.resample()`.

Parameters

- **data** (*~PandasObj*) – a DataFrame or Series with a datetime index
- **freq** (`Union[int, str]`) – resampling frequency in *pandas* notation
- **hour** (`int`) – reference hour; all datetimes in the new index will have this hour
- **preserve_ints** (`bool`) – after interpolation, columns containing integers in the original dataframe are rounded and converted back to int
- **method** – interpolation method (see `pandas.DataFrame.interpolate()`)
- ****interpolation** – other interpolation keyword argument different from *method* passed to `pandas.DataFrame.interpolate()`

Return type ~PandasObj

Returns a new Dataframe/Series with index elements spaced according to `freq`

See also:

reindex_by_interpolating()

2.1.6 iccas.queries module

iccas.queries.**age_grouper**(*cuts*, *fmt_last='>={}'*)

Return type `Dict[str, str]`

iccas.queries.**aggregate_age_groups**(*counts*, *cuts*, *fmt_last='>={}'*)

Aggregates counts for different age groups summing them together.

Parameters

- **counts** (*~PandasObj*) – can be a Series with age groups as index or a DataFrame with age groups as columns, either in a simple Index or in a MultiIndex (no matter in what level)
- **cuts** (`Union[int, Sequence[int]]`) – a single integer N means "cut each N years"; a sequence of integers determines the start ages of new age groups; 0 is implicitly the start age of the first group, even if not present in `cuts`.
- **fmt_last** (`str`) – format string for the last "unbounded" age group

Return type ~PandasObj

Returns A Series/DataFrame with the same "structure" of the input but with aggregated age groups.

iccas.queries.**average_by_period**(*counts*, *freq*)
Returns a new Series/DataFrame with average counts (cases/deaths) by period (e.g. months, weeks, n days ecc)

Parameters

- **counts** (*~PandasObj*) –
- **freq** (Union[str, int]) – period frequency parameter (whatever accepted by pandas)

Returns:

Return type ~PandasObj

iccas.queries.**cols**(*prefixes*, *fields='*'*)
Generates a list of columns by combining prefixes with fields.

Parameters

- **prefixes** (str) – string containing one or multiple of the following characters: - 'm' for males - 'f' for females - 't' for totals (no prefix) - '*' for all
- **fields** (Union[str, Sequence[str]]) – values: 'cases', 'deaths', 'cases_percentage', 'deaths_percentage', 'fatality_rate', '*'

Return type List[str]

Returns a list of string

iccas.queries.**count_by_period**(*counts*, *freq*)
Returns a new Series/DataFrame with counts (cases/deaths) by period (e.g. months, weeks, n days ecc)

Parameters

- **counts** (*~PandasObj*) –
- **freq** (Union[str, int]) – period frequency parameter (whatever accepted by pandas)

Returns:

Return type ~PandasObj

iccas.queries.**fatality_rate**(*counts*, *shift*)
Computes the fatality rate as a ratio between the total number of deaths and the total number of cases shift days before.

counts is resampled with interpolation if needed.

iccas.queries.**get_unknown_sex_count**(*counts*, *variable*)
Returns cases/deaths of unknown sex for each age group

Return type DataFrame

iccas.queries.**only_cases**(*data*)
Returns only columns ['cases', 'female_cases', 'male_cases']

Return type DataFrame

iccas.queries.**only_counts**(*data*)
Returns only cases and deaths columns (including sex-specific columns), dropping all other columns that are computable from these.

Return type DataFrame

iccas.queries.**only_deaths**(*data*)
Returns only columns ['deaths', 'female_deaths', 'male_deaths']

Return type DataFrame

iccas.queries.**product_join**(**string_iterables*, *sep=''*)

Return type Iterable[str]

iccas.queries.**running_average**(*counts*, *window*, *step=1*, ***resample_kwargs*)

Given counts for cases/deaths, returns the average daily number of new cases/deaths inside a temporal window of window, moving the window step days a time.

Parameters

- **counts** (*~PandasObj*) –
- **window** (int) –
- **step** (int) –

Returns:

Return type ~PandasObj

iccas.queries.**running_count**(*counts*, *window*, *step=1*, ***resample_kwargs*)

Given counts for cases and/or deaths, returns the number of new cases inside a temporal window of window days that moves forward by steps of step days.

Parameters

- **counts** (*~PandasObj*) –
- **window** (int) –
- **step** (int) –

Returns:

Return type ~PandasObj

2.1.7 iccas.types module

2.1.8 Module contents

iccas.**age_grouper**(*cuts*, *fmt_last='>={}'*)

Return type Dict[str, str]

iccas.**aggregate_age_groups**(*counts*, *cuts*, *fmt_last='>={}'*)

Aggregates counts for different age groups summing them together.

Parameters

- **counts** (*~PandasObj*) – can be a Series with age groups as index or a DataFrame with age groups as columns, either in a simple Index or in a MultiIndex (no matter in what level)
- **cuts** (Union[int, Sequence[int]]) – a single integer N means "cut each N years"; a sequence of integers determines the start ages of new age groups; 0 is implicitly the start age of the first group, even if not present in cuts.
- **fmt_last** (str) – format string for the last "unbounded" age group

Return type ~PandasObj

Returns A Series/DataFrame with the same "structure" of the input but with aggregated age groups.

iccas.**average_by_period**(*counts*, *freq*)

Returns a new Series/DataFrame with average counts (cases/deaths) by period (e.g. months, weeks, n days ecc)

Parameters

- **counts** (*~PandasObj*) –
- **freq** (Union[str, int]) – period frequency parameter (whatever accepted by pandas)

Returns:

Return type ~PandasObj

iccas.**cols**(*prefixes*, *fields='*'*)

Generates a list of columns by combining prefixes with fields.

Parameters

- **prefixes** (str) – string containing one or multiple of the following characters: - 'm' for males - 'f' for females - 't' for totals (no prefix) - '*' for all
- **fields** (Union[str, Sequence[str]]) – values: 'cases', 'deaths', 'cases_percentage', 'deaths_percentage', 'fatality_rate', '*'

Return type List[str]

Returns a list of string

iccas.**count_by_period**(*counts*, *freq*)

Returns a new Series/DataFrame with counts (cases/deaths) by period (e.g. months, weeks, n days ecc)

Parameters

- **counts** (*~PandasObj*) –
- **freq** (Union[str, int]) – period frequency parameter (whatever accepted by pandas)

Returns:

Return type ~PandasObj

iccas.**fatality_rate**(*counts*, *shift*)

Computes the fatality rate as a ratio between the total number of deaths and the total number of cases shift days before.

counts is resampled with interpolation if needed.

iccas.**fix_monotonicity**(*data*, *method='pchip'*, ***interpolation*)

Replaces tracts of all cases and deaths time series that break the non-decreasing trend of the series with interpolated data. This function also ensures that the following conditions are still satisfied even after the "correction":

```
male_cases + female_cases <= cases
male_deaths + female_deaths <= deaths
```

Non-integer columns, if present, are ignored and returned as they are in the output DataFrame.

Parameters

- **data** (DataFrame) – a DataFrame containing all integer columns about cases and deaths
- **method** – interpolation method

Returns a DataFrame with all integer time series (columns) modified so that they are non-decreasing time series

iccas.**get**(*cache_dir=PosixPath('/home/docs/.iccas')*)

Returns the latest version of the ICCAS dataset in a pandas.DataFrame (as it's returned by *load()*).

This function uses RemoteFolderCache.get(), which caches.

Raises

- **request.exceptions.ConnectionError** – if the server is unreachable
- **and no dataset is available in cache_dir** –

Return type DataFrame

iccas.**get_by_date**(*date*, *keep_date=False*, *cache_dir=PosixPath('/home/docs/.iccas')*)

Return type Tuple[DataFrame, Timestamp]

iccas.**get_population_by_age**(*cache_dir=PosixPath('/home/docs/.iccas')*)
Returns a DataFrame with "age" as index and two columns: "value" (absolute counts) and "percentage" (<=1.0)

Return type DataFrame

iccas.**get_population_by_age_group**(*cache_dir=PosixPath('/home/docs/.iccas')*)
Returns a DataFrame with "age_group" as index and two columns: "value" (absolute counts) and "percentage" (<=1.0)

Return type DataFrame

iccas.**get_unknown_sex_count**(*counts*, *variable*)
Returns cases/deaths of unknown sex for each age group

Return type DataFrame

iccas.**get_url**(*date=None*, *fmt='csv'*)
Returns the url of a dataset in a given format. If *date* is None, returns the URL of the full dataset.

Return type str

iccas.**language**(*lang*)

iccas.**load**(*path*)

Return type DataFrame

iccas.**load_single_date**(*path*, *keep_date=False*)
Loads a dataset containing data for a single date.

By default (*keep_date=False*), the *date* column is dropped and the datetime is stored in the *attrs* of the DataFrame. If instead *keep_date=True*, the returned dataset has a MultiIndex *(date, age_group)*.

Parameters

- **path** (Union[str, Path]) –
- **keep_date** (bool) – whether to drop the date column (containing a single datetime value)

Return type DataFrame

iccas.**only_cases**(*data*)
Returns only columns ['cases', 'female_cases', 'male_cases']

Return type DataFrame

iccas.**only_counts**(*data*)
Returns only cases and deaths columns (including sex-specific columns), dropping all other columns that are computable from these.

Return type DataFrame

iccas.**only_deaths**(*data*)
Returns only columns ['deaths', 'female_deaths', 'male_deaths']

Return type DataFrame

iccas.**reindex_by_interpolating**(*data*, *new_index*, *preserve_ints=True*, *method='pchip'*, ***interpolation*)

Reindexes *data* and fills new values by interpolation (PCHIP, by default).

This function was motivated by the fact that `pandas.DataFrame.resample()` followed by `pandas.DataFrame.resample()` doesn't take into account misaligned datetimes.

Parameters

- **data** (*~PandasObj*) – a DataFrame or Series with a datetime index
- **new_index** (`DatetimeIndex`) –
- **preserve_ints** (`bool`) – after interpolation, columns containing integers in the original dataframe are rounded and converted back to int
- **method** – interpolation method (see `pandas.DataFrame.interpolate()`)
- ****interpolation** – other interpolation keyword argument different from *method* passed to `pandas.DataFrame.interpolate()`

Return type ~PandasObj

Returns a new Dataframe/Series

See also:

reindex_by_interpolating()

iccas.**resample**(*data*, *freq='1D'*, *hour=18*, *preserve_ints=True*, *method='pchip'*, ***interpolation*)

Resamples *data* and fills missing values by interpolation.

The resulting index is a *pandas.DatetimeIndex* whose elements are spaced accordingly to *freq* and having the time set to *{hour}:00*.

In the case of "day frequencies" ('{num}D'), the index always includes the latest date (*data.index[-1]*): the new index is a datetime range built going backwards from the latest date.

This function was motivated by the fact that `pandas.DataFrame.resample()` followed by `pandas.DataFrame.resample()` doesn't take into account misaligned datetimes. If you want to back-fill or forward-fill, just use `DataFrame.resample()`.

Parameters

- **data** (*~PandasObj*) – a DataFrame or Series with a datetime index
- **freq** (`Union[int, str]`) – resampling frequency in *pandas* notation
- **hour** (`int`) – reference hour; all datetimes in the new index will have this hour
- **preserve_ints** (`bool`) – after interpolation, columns containing integers in the original dataframe are rounded and converted back to int
- **method** – interpolation method (see `pandas.DataFrame.interpolate()`)
- ****interpolation** – other interpolation keyword argument different from *method* passed to `pandas.DataFrame.interpolate()`

Return type ~PandasObj

Returns a new Dataframe/Series with index elements spaced according to `freq`

See also:

reindex_by_interpolating()

iccas.**running_average**(*counts*, *window*, *step=1*, ***resample_kwargs*)

Given counts for cases/deaths, returns the average daily number of new cases/deaths inside a temporal window of `window`, moving the window `step` days a time.

Parameters

- **counts** (*~PandasObj*) –
- **window** (int) –
- **step** (int) –

Returns:

Return type ~PandasObj

iccas.**running_count**(*counts*, *window*, *step=1*, ***resample_kwargs*)

Given counts for cases and/or deaths, returns the number of new cases inside a temporal window of `window` days that moves forward by steps of `step` days.

Parameters

- **counts** (*~PandasObj*) –
- **window** (int) –
- **step** (int) –

Returns:

Return type ~PandasObj

iccas.**set_language**(*lang*)

Sets the language. Supported languages: Italian (“it”) and English (“en”)

iccas.**set_locale**(*lang*)

Apart from setting the internal language of the package, also sets the locale accordingly so that pandas/matplotlib displays translated dates

CHAPTER

THREE

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:

3.1 Types of Contributions

3.1.1 Report Bugs

Report bugs at https://github.com/janLuke/iccas-python/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.

3.1.2 Fix Bugs

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

3.1.3 Implement Features

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

3.1.4 Write Documentation

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

3.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/janLuke/iccas-python/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 :)

3.2 Get Started!

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

1. Fork the *iccas-python* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/iccas-python.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 iccas
$ cd iccas/
$ 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 iccas 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.

3.3 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/janLuke/iccas/pull_requests and make sure that the tests pass for all supported Python versions.

3.4 Tips

To run a subset of tests:

```
$ pytest tests.test_iccas
```

3.5 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

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

L

M

N

O

P

R

S

T