
reportsrender

Gregor Sturm

Dec 13, 2019

CONTENTS:

1	Generate reproducible reports from Rmarkdown or jupyter notebooks	1
2	Getting started	3
3	Usage from command line	5
4	Installation	7
4.1	Conda (recommended):	7
4.2	Manual installation:	7
5	Features	9
5.1	Execution engines	9
5.2	Supported notebook formats	9
5.3	Hiding cell inputs/outputs	9
5.4	Parametrized notebooks	10
5.5	Sharing reports	11
5.6	Combine notebooks into a pipeline	11
6	Usage as Python library	13
6.1	reportsrender.render_rmd	13
6.2	reportsrender.render_papermill	13
6.3	reportsrender.run_pandoc	14
6.4	reportsrender.build_index	14
	Python Module Index	15
	Index	17

GENERATE REPRODUCIBLE REPORTS FROM RMARKDOWN OR JUPYTER NOTEBOOKS

Reportsrender allows to create reproducible, consistently looking HTML reports from both jupyter notebooks and Rmarkdown files. It makes use of [papermill](#) and [Rmarkdown](#) to execute notebooks and uses [Pandoc](#) to convert them to HTML.

Features:

- two execution engines: [papermill](#) and [Rmarkdown](#).
- support any format supported by [jupyter](#).
- create self-contained HTML that can be shared easily.
- hide inputs and/or outputs of cells.
- parametrized reports

See the [documentation](#) for more details!

GETTING STARTED

- Execute an rmarkdown document to HTML using the Rmarkdown engine

```
reportsrender --engine=rmd my_notebook.Rmd report.html
```

- Execute a parametrized jupyter notebook with papermill

```
reportsrender --engine=papermill jupyter_notebook.ipynb report.html --params="data_  
↪file=table.tsv"
```


USAGE FROM COMMAND LINE

```
reportsrender

Execute and render a jupyter/Rmarkdown notebook.
The `index` subcommand generates an index html
or markdown file that links to html documents.

Usage:
  reportsrender <notebook> <out_file> [--cpus=<cpus>] [--params=<params>] [--engine=
↪<engine>]
  reportsrender index [--index=<index_file>] [--title=<title>] [--] <html_files>...
  reportsrender --help

Arguments and options:
  <notebook>          Input notebook to be executed. Can be any format supported by ↪
↪jupyter.
  <out_file>          Output HTML file.
  -h --help           Show this screen.
  --cpus=<cpus>      Number of CPUs to use for Numba/Numpy/OpenBLAS/MKL [default: ↪
↪1]
  --params=<params>  space-separated list of key-value pairs that will be passed
to papermill/Rmarkdown.
E.g. "input_file=dir/foo.txt output_file=dir2/bar.html"
  --engine=<engine>  Engine to execute the notebook. [default: auto]

Arguments and options of the `index` subcommand:
  <html_files>       List of HTML files that will be included in the index. The ↪
↪tool
will generate relative links from the index file to these ↪
↪files.
  --index=<index_file> Path to the index file that will be generated. Will be
overwritten if exists. Will auto-detect markdown (.md) and
HTML (.html) format based on the extension. [default: index.
↪html]
  --title=<title>     Headline of the index. [default: Index]

Possible engines are:
  auto              Use `rmd` engine for `*.Rmd` files, papermill otherwise.
  rmd               Use `rmarkdown` to execute the notebook. Supports R and
python (through reticulate)
  papermill         Use `papermill` to execute the notebook. Works for every
kernel available in the jupyter installation.
```


INSTALLATION

4.1 Conda (recommended):

As this reportsrender depends on both R and Python packages, I recommend to install it through [conda](#). The following command will install reportsrender and all its dependencies in the current conda environment:

```
conda install -c conda-forge grst::reportsrender
```

If you prefer not to use conda, you can follow the approach below:

4.2 Manual installation:

4.2.1 Get dependencies:

- Python
- [pandoc](#)

For the Rmarkdown render engine additionally (there is no need to install them if you are not going to use the Rmarkdown rendering engine):

- R and the following packages:

```
rmarkdown  
reticulate
```

then,

4.2.2 Install from pip:

```
pip install reportsrender
```

or,

4.2.3 Install from github:

```
pip install flit  
flit installfrom github:grst/reportsrender
```


FEATURES

5.1 Execution engines

Reportsrender comes with two execution engines:

- **Rmarkdown.** This engine makes use of the [Rmarkdown package](#) implemented in R. Essentially, this engine calls `Rscript -e "rmarkdown::render()"`. It supports Rmarkdown notebooks (*Rmd* format) and python notebooks through [reticulate](#).
- **Papermill.** This engine combines [papermill](#) and [nbconvert](#) to parametrize and execute notebooks. It supports any programming language for which a jupyter kernel is installed.

5.2 Supported notebook formats

Reportsrender uses [jupyter](#) to convert between input formats. Here is the full list of [supported formats](#).

So no matter if you want to run an *Rmd* file with papermill, an *ipynb* with Rmarkdown or a [Hydrogen percent script](#), reportsrender has got you covered.

5.3 Hiding cell inputs/outputs

You can hide inputs and or outputs of individual cells:

5.3.1 Papermill engine:

Within a jupyter notebook:

- edit cell metadata
- add one of the following *tags*: `hide_input`, `hide_output`, `remove_cell`

```
{
  "tags": [
    "remove_cell"
  ]
}
```

5.3.2 Rmarkdown engine:

- all native input control options (e.g. `results='hide'`, `include=FALSE`, `echo=FALSE`) are supported. See the [Rmarkdown documentation](#) for more details.

Jupyter automatically converts the tags to Rmarkdown options for all supported formats.

5.4 Parametrized notebooks

5.4.1 Papermill engine:

- See the [Papermill documentation](#)

Example:

- Add the tag `parameters` to the metadata of a cell in a jupyter notebook.
- Declare default parameters in that cell:

```
input_file = '/path/to/default_file.csv'
```

- Use the variable as any other:

```
import pandas as pd
pd.read_csv(input_file)
```

5.4.2 Rmarkdown engine:

- See the [documentation](#).

Example:

- Declare the parameter in the yaml frontmatter.
- You can set default parameters that will be used when the notebook is executed interactively in Rstudio. They will be overwritten when running through `reportsrender`.

```
---
title: My Document
output: html_document
params:

  input_file: '/path/to/default_file.csv'
---
```

- Access the parameters from the code:

```
read_csv(params$input_file)
```

5.4.3 Be compatible with both engines:

Yes it's possible! You can execute the same notebook with both engines. Adding parameters is a bit more cumbersome though.

Example (Python notebook stored as `.Rmd` file using `jupyter`):

```

---
title: My Document
output: html_document
params:
  input_file: '/path/to/default_file.csv'
---

```{python tags=c("parameters")}
try:
 # try to get param from Rmarkdown using reticulate.
 input_file = r.params["input_file"]
except:
 # won't work if running papermill. Re-declare default parameters.
 input_file = "/path/to/default_file.csv"
```

```

5.5 Sharing reports

Reportsrender create self-contained HTML files that can be easily shared, e.g. via email.

I do, however, recommend using [github pages](#) to upload and share your reports. A central website serves as a *single point of truth* and eliminates the problem of different versions of your reports being emailed around.

You can make use of *reportsrender index* to automatically generate an index page listing multiple reports:

Say, you generated several reports and already put them into your github-pages directory:

```

gh-pages
├── 01_preprocess_data.html
├── 02_analyze_data.html
└── 03_visualize_data.htmlp

```

Then you can generate an index file listing and linking to your reports by running

```
reportsrender index --index gh-pages/index.md gh-pages/*.html
```

For more details see *Usage from command line* and *reportsrender.build_index()*

5.5.1 Password protection

Not all analyses can be shared publicly. Unfortunately, github-pages does not support password protection.

There is a [workaround](#), though:

As github-pages doesn't list directories, you can simply create a long, cryptic subdirectory, e.g. *t8rry6poj7ua6eujqpb57* and put your reports within. Only people with whom you share the exact link will be able to access the site.

5.6 Combine notebooks into a pipeline

Reportsrender is built with pipelines in mind. You can easily combine individual analysis steps into a fully reproducible pipeline using workflow engines such as [Nextflow](#) or [Snakemake](#).

A full example how such a pipeline might look like is available in a dedicated GitHub repository: [universal_analysis_pipeline](#). It's based on Nextflow, but could easily be adapted to other pipelining engines.

USAGE AS PYTHON LIBRARY

Reportsrender provides a public API that can be used to execute and convert notebooks to HTML:

Execute and render notebooks as HTML reports.

| | |
|---|---|
| <code>render_rmd(input_file, output_file[, params])</code> | Wrapper function to render an Rmarkdown document with the R <i>rmarkdown</i> package and convert it to HTML using pandoc and a custom template. |
| <code>render_papermill(input_file, output_file[, ...])</code> | Wrapper function to render a jupyter/jupyter notebook with papermill and pandoc. |
| <code>run_pandoc(in_file, out_file[, res_path, ...])</code> | Convert to HTML using pandoc. |
| <code>build_index(html_files, output_file[, title])</code> | Create an index file referencing all specified html files. |

6.1 reportsrender.render_rmd

`reportsrender.render_rmd(input_file, output_file, params=None)`

Wrapper function to render an Rmarkdown document with the R *rmarkdown* package and convert it to HTML using pandoc and a custom template.

Parameters

- **input_file** (`str`) – path to input (Rmd) file
- **output_file** (`str`) – path to output (html) file
- **params** (`Optional[dict]`) – Dictionary that will be passed to *params* arg of *rmarkdown::render*. See <https://bookdown.org/yihui/rmarkdown/parameterized-reports.html> for more details.

6.2 reportsrender.render_papermill

`reportsrender.render_papermill(input_file, output_file, params=None)`

Wrapper function to render a jupyter/jupyter notebook with papermill and pandoc.

Parameters

- **input_file** (`str`) – path to input file. Can be any format supported by jupyter.
- **output_file** (`str`) – path to output (html) file.
- **params** (`Optional[dict]`) – parameter dictionary that will be passed to papermill. See <https://papermill.readthedocs.io/en/latest/usage-parameterize.html> for more details.

6.3 reportsrender.run_pandoc

`reportsrender.run_pandoc` (*in_file*, *out_file*, *res_path=None*, *template_file=None*, *css_file=None*)

Convert to HTML using pandoc.

Will create a standalone, self-contained html based on the specified template.

Parameters

- **in_file** (`str`) – path to input file. Can be any format supported by pandoc. The format will be inferred from the file extension.
- **out_file** (`str`) – path to output (html) file.
- **res_path** (`Optional[Collection[str]]`) – List of pandoc resource paths (pandoc will look here for asset files). If no `template_file` is provided the resource path of the default template will be appended.
- **template_file** (`Optional[str]`) – path to the pandoc template. Per default, the *adaptive-bootstrap* template shipped with this package will be used.
- **css_file** (`Optional[str]`) – path to the css file used by pandoc. Per default, the css file from the *adaptive-bootstrap* template shipped with this package will be used.

6.4 reportsrender.build_index

`reportsrender.build_index` (*html_files*, *output_file*, *title='Index'*)

Create an index file referencing all specified html files.

Parameters

- **html_files** (`List[str]`) – List of documents to include in the index. The items will appear in the same order as in the list.
- **output_file** (`str`) – Path to output file. Can either end with *.md* or *.html*. In case of *.html* pandoc will be ran to convert the markdown file to HTML.
- **title** (`str`) – H1-title of the page

PYTHON MODULE INDEX

r

reportsrender, 13

INDEX

B

`build_index()` (*in module reportsrender*), 14

R

`render_papermill()` (*in module reportsrender*), 13

`render_rmd()` (*in module reportsrender*), 13

`reportsrender` (*module*), 13

`run_pandoc()` (*in module reportsrender*), 14