nbdocs Documentation

Release 1.0

Roman Valov

Nov 25, 2020

CONTENTS:

CHAPTER

ONE

TASK 1

Hi, Erik. In this document I will guide you step-by-step into process of creating and deploying docs example to RTD. As you've requested the docs will have Jupyter Notebook.

1.1 Step 1. Overview

The RTD service is a document-hosting service for GitHub-hosted projects. It's free and it's only supposed to host documentation projects, not custom sites or files.

RTD is built on top of widely-used Sphinx project (https://www.sphinx-doc.org/en/master/). Sphinx is documentation generator, it produces documentation in various formats (primarily html) from templates files.

1.2 Step 2. Templates

Sphinx templates are just text files formatted using RST markup language. RST is similar to Markdown (markup language used in GitHub readme files or StackOverflow posts).

To get the idea of RST and learn basic constructs please read following doc:

https://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

It's actually possible to use Markdown to write tempates for Sphinx however I've never used this possibility before.

1.3 Step 3. Installation

At first you have to install the Sphinx document generator on your system. If you're using Ubuntu run:

```
sudo apt-get install python3-sphinx
```

Alternatively you could use `pip3` command to install `sphinx` package from PyPI repository:

```
sudo pip3 install sphinx
```

Please note that there is python2 and python3 version of Sphinx could be available in your repository. You should prefer the same version as your primary project uses (Sphinx analyzes your python code to build documentation). Also please ensure you have only one version installed because you could get conflicts in ohter case.

Once installation completed, go to your source code directory and run:

```
sphinx-quickstart
```

This wizard will ask you several questions about your project and configuration options. It's safe to keep with defaults for all the configuration options, just ensure to set proper project name, author and version.

Take a minute to consider the structure of your sources. You should keep your *.ipynb files in the root of documentation.

1.4 Step 4. Build docs

When wizard completes you should find following files in your repository:

```
conf.py (configuration)
index.rst (home page template)
Makefile (build scripts for Linux)
make.bat (build scripts for Windows)
```

Directory could contain other files and subdirectories but they're not importrant.

In order to build html version of your docs run following command:

```
make html
```

This command will generate html docs in `_build/html` subdirectory (if you haven't changed the defaults) Now you can navigate with your file explorer to this directory and open `index.html` file in your browser. If you're on Ubuntu you could run following command without leaving the console:

```
xdg-open _build/html/index.html
```

1.5 Step 5. Notebooks extension

Hope everything is working fine and you were able to see your documentation stub in the browser. You could freely try various formatting constructs according to previously mentioned RST primer. But for the sake of simplicity I will continue to show you how to integrate Jupyter Notebooks.

Sphinx is extensible software and in order to render Jupyter Notebooks you have to install `nbsphinx` extension.

If you're on Ubuntu you should run:

```
sudo apt-get install python3-nbsphinx
```

Alternatively you could install PyPI version of the package:

```
sudo pip3 install nbsphinx
```

Now you should enable the extension in the `conf.py`. Open the file with your favorite editor and find `extensions` stanza. Add `'nbsphinx'` to the list of extensions, i.e.:

```
extensions = ['nbsphinx']
```

1.6 Step 6. Rendering notebooks

Now it's time to add your notebook to the docs. Make sure you have you *.ipynb file in the same directory with `index.rst`. Open `index.rst` with your favorite editor. By default the auto-generated file has Table of Contents and several standard links. You should modify table of contents and add the name of your *.ipynb file to the list. The .ipynb extension should be omitted. Also make sure your entry is idented on the same level as colon-marked stanzas:

```
.. toctree::
    :maxdepth: 2
    :caption: Contents:

    maps
```

As you see in my case I've added `maps` entry to the list. It's actually a copy of `python_maps_example.ipynb` from your repository renamed for the sake of convenience.

Once ready please run the build again and check results in your browser. Based on the `maps` file contents found in your repository you will get index page with pair of links on it. Each of the links will point to the sub-section in newly created `maps.html` file built from your notebook.

The same way you could freely use arbitrary *.ipynb file instead of RST-file, even instead of `index.rst`. However you have to delete `index.rst` file in latter case because *.rst files are prioritized by Sphinx.

1.7 Step 7. Moving to GitHub

If everything is working fine locally it's time to move to public hosting. In order to do that you should commit and upload your files to your GitHub repository.

The following files should be commited and pushed to the repository:

```
index.rst <your-noteook-file>.ipynb conf.py
```

As of `Makefile` and `make.bat` – they're just convenient wrappers for local builds and not required for RTD.

You could check that GitHub will render not only *.ipynb files in it's web-interface, but also *.rst files.

1.8 Step 8. Read the Docs

When your files are available on GitHub it's time to register an account on ReadTheDocs and link your GitHub repository.

Go to https://readthedocs.org/accounts/login/ and press the `Sign in with GitHub` button.

In the profile page of ReadTheDocs you will find `Import project` button, use it and select your repository from the list.

Once imported all the machinery should be set up by ReadTheDocs to start build and set up rebuild on each commit to your repo.

Please take a time to get familiar with ReadTheDocs interface.

In general it's usefull to be able to check the status of the last build and view the build logs.

1.9 Step 9. Notebooks on Read the Docs

By default ReadTheDocs is not configured to use Notebooks extension previously used for local build.

In order to change the limitations you have to add pair of configuration files to your repository.

At first, add the `requirements.txt` file to the same dir where you have `index.rst` located and add following lines:

```
ipykernel
nbsphinx
```

These lines will instruct ReadTheDocs build to download packages from the PyPI archive.

On your local setup `ipykernel` is usually installed as a dependency for Jupyter and `nbsphinx` was installed as a part of the tutorial.

At second, you have to add configuration file for ReadTheDocs service itself which relies on the `requirements.txt` defined. Configuration file for ReadTheDocs should be named `.readthedocs.yml` and should be located in top dir of your repository:

```
version: 2
formats: all
python:
  version: 3
  install:
  - requirements: docs/requirements.txt
  system_packages: true
```

As you see in my case the version of Python interpreter is set to 3 and `requirements.txt` is located in `docs` subdir.

Once files added do a commit and push to your repository, the ReadTheDocs will do the rebuild in a while.

1.10 Step 10. Themming

Sphinx supports themming. In my case Sphinx tools bundled with the distro are patched to use Alabaster theme by default.

In order to force your documentation pages to use particular theme it should be configured via `html_theme` parameter.

For example to use default ReadTheDocs theme you have to set `html_theme='sphinx_rtd_theme'` in your configuration file.

Being default for ReadTheDocs service it will be handled automatically on ReadTheDocs service. However if you wish to give it a try locally you have to install theme's python package:

```
sudo pip3 install sphinx-rtd-theme
```

[29]:
```
import numpy as np
import cartopy

import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
```

CHAPTER

TWO

POINT DATA

[32]:
```
import pandas as pd
```

[2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/2011_
↪february_us_airport_traffic.csv')
```

[36]:
```
df.head()
```

[36]:
```
  iata                            airport               city state country  \
0  ORD       Chicago O'Hare International            Chicago    IL     USA
1  ATL  William B Hartsfield-Atlanta Intl            Atlanta    GA     USA
2  DFW    Dallas-Fort Worth International  Dallas-Fort Worth    TX     USA
3  PHX   Phoenix Sky Harbor International            Phoenix    AZ     USA
4  DEN                        Denver Intl             Denver    CO     USA

         lat        long    cnt
0  41.979595  -87.904464  25129
1  33.640444  -84.426944  21925
2  32.895951  -97.037200  20662
3  33.434167 -112.008056  17290
4  39.858408 -104.667002  13781
```

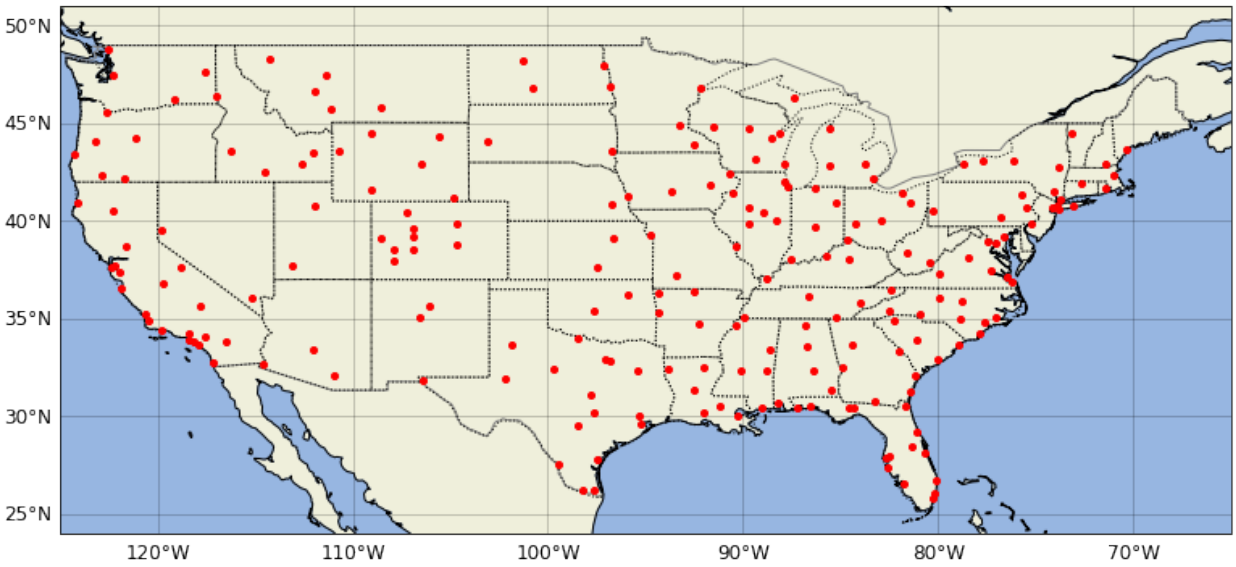
[35]:
```
plt.figure(figsize=(13,6.2))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())

# Add features
ax.add_feature(cartopy.feature.LAND)
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')

# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
```

(continues on next page)

(continued from previous page)

```
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

# Add airport locations
ax.plot(df.long, df.lat, transform=cartopy.crs.PlateCarree(),marker='o', color='red',␣
↪markersize=4, linestyle='')

plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/cartopy/mpl/style.py:76: UserWarning:␣
↪facecolor will have no effect as it has been defined as "never".
  warnings.warn('facecolor will have no effect as it has been '
```

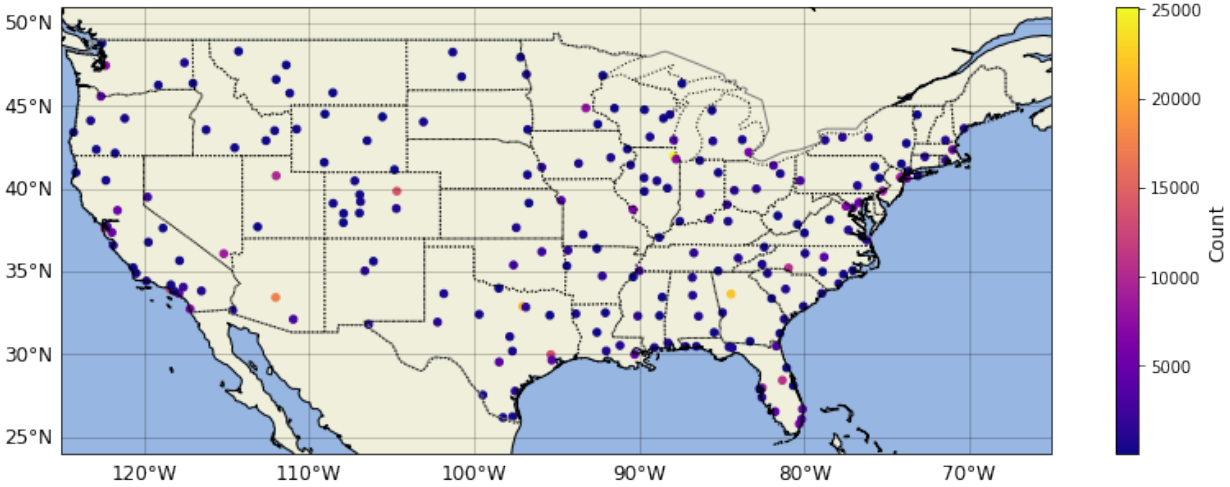


[43]:
```
fig = plt.figure(figsize=(14,5))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())

# Add features
ax.add_feature(cartopy.feature.LAND)
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')
```

(continues on next page)

(continued from previous page)

```
# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

# Add airport locations with color showing number of arrivals
p = ax.scatter(df.long, df.lat, c=df.cnt, transform=cartopy.crs.PlateCarree(), s = 20,
↪ cmap = 'plasma')
cb = fig.colorbar(p)
cb.set_label(r'Count',fontsize=12)
plt.show()
```



CHAPTER

THREE

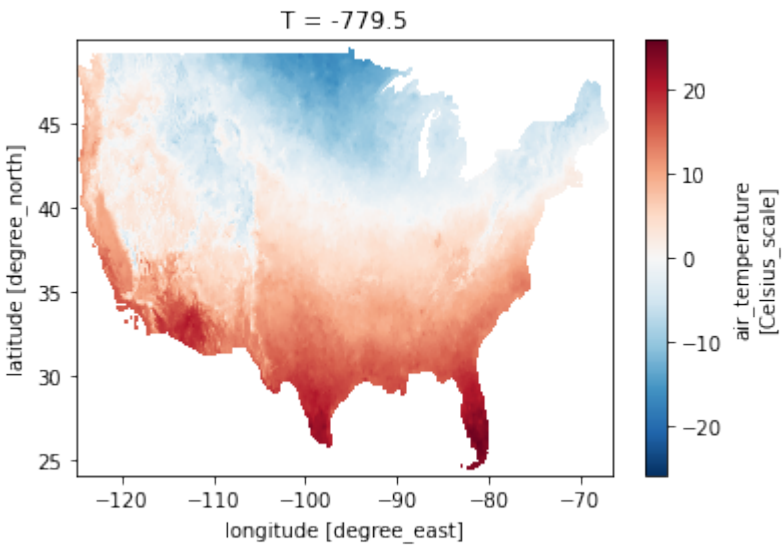
GRIDDED DATA

[53]:
```
import xarray as xr
```

[56]:
```
ds = xr.open_dataset("http://iridl.ldeo.columbia.edu/SOURCES/.OSU/.PRISM/.monthly/dods
↪",decode_times=False)
```

[59]:
```
ds.tmax[0].plot()
```

[59]:
```
<matplotlib.collections.QuadMesh at 0x7f2c0c9a51d0>
```



[71]:
```
dx = np.unique(np.round(np.diff(ds.X),4))[0]
dy = -np.unique(np.round(np.diff(ds.Y),4))[0]
print('Grid spacing\n',dx,dy)
```

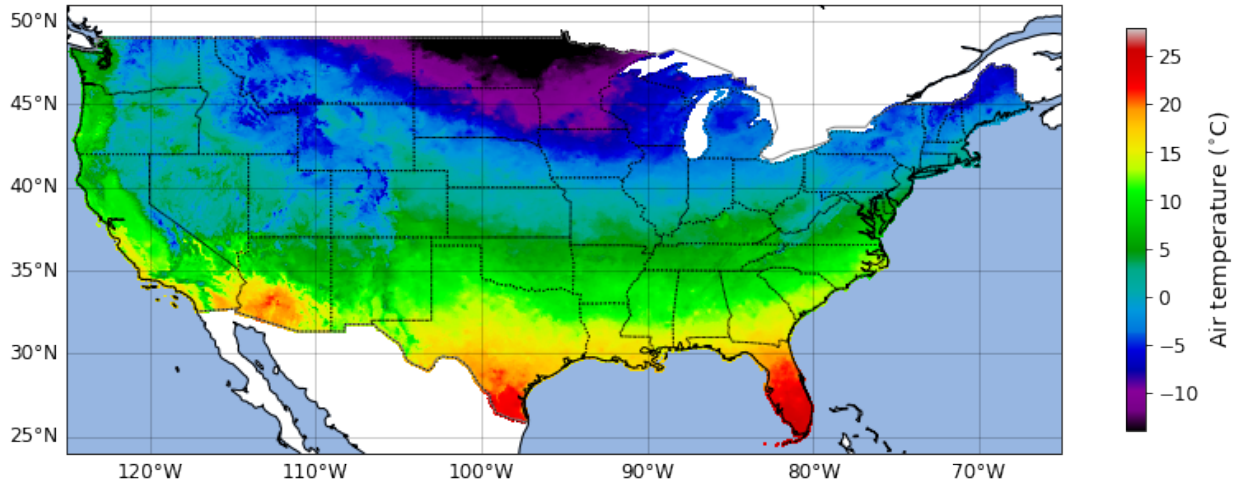
```
Grid spacing
 0.0417 0.0417
```

[75]:
```
fig = plt.figure(figsize=(14,5))

ax = plt.axes(projection=cartopy.crs.PlateCarree())

# Set lat/lon limit of map
ax.set_extent([-125, -65, 24, 51], crs=cartopy.crs.PlateCarree())
```

(continues on next page)

(continued from previous page)

```
# Add features
ax.add_feature(cartopy.feature.OCEAN)
ax.add_feature(cartopy.feature.COASTLINE)
ax.add_feature(cartopy.feature.BORDERS, linestyle='-', color='grey')
ax.add_geometries(cartopy.io.shapereader.Reader(cartopy.io.shapereader.natural_earth\
                                                (resolution='110m',category='cultural
↪',
                                                 name='admin_1_states_provinces_lakes_
↪shp')).geometries(),
                  cartopy.crs.PlateCarree(),facecolor='none',edgecolor='black',ls=':')

# Add lat/lon grid
gl = ax.gridlines(cartopy.crs.PlateCarree(), draw_labels=True, linewidth=1.0,␣
↪linestyle='-', color='k',alpha=0.2)
gl.xlocator = mticker.FixedLocator(np.arange(-120,-60,10))
gl.ylocator = mticker.FixedLocator(np.arange(25,55,5))
gl.top_labels = False
gl.right_labels = False
gl.xlabel_style= {'size': 12, 'color': 'k'}
gl.ylabel_style= {'size': 12, 'color': 'k'}

p = ax.pcolormesh(ds.X-dx/2, ds.Y-dy/2, np.ma.masked_invalid(ds.tmax[0]),
                  cmap='nipy_spectral', vmin=-14, vmax=28,transform=cartopy.crs.
↪PlateCarree())

cb = fig.colorbar(p, orientation='vertical', shrink=0.9, ticks=np.arange(-10,30,5))
cb.ax.tick_params(labelsize=12)
cb.set_label(r'Air temperature ($^{\circ}$C)', fontsize=14)
plt.show()
```



[ ]:

CHAPTER

FOUR

CITATIONS USING NOTEBOOK HTML CITATIONS

Jan-Erik Tesdal[1]**, Ryan Abernathey**[1] **and Ian Fenty**[2]

Mentioning the article [3].

This section demonstrates the closure of the global heat budget in ECCOv4. The steps and Python code has been directly derived from the calculations and MATLAB code in *"A Note on Practical Evaluation of Budgets in ECCO Version 4 Release 3"* by Christopher G. Piecuch (https://ecco.jpl.nasa.gov/drive/files/Version4/Release3/doc/v4r3_budgets_howto.pdf).

CHAPTER

FIVE

CITATIONS USING MARKDOWN FOOTNOTES SYNTAX

Jan-Erik Tesdal[1][2], **Ryan Abernathey**[3] **and Ian Fenty**[4]

This section demonstrates the closure of the global heat budget in ECCOv4. The steps and Python code has been directly derived from the calculations and MATLAB code in *"A Note on Practical Evaluation of Budgets in ECCO Version 4 Release 3"* by Christopher G. Piecuch (https://ecco.jpl.nasa.gov/drive/files/Version4/Release3/doc/v4r3_budgets_howto.pdf).

[1] Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY, USA
[2] *Corresponding author*: jt2796@columbia.edu
[3] Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY, USA
[4] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA

CHAPTER

SIX

CITATIONS USING MARKDOWN SUPERSCRIPT SYNTAX (PANDOC ONLY)

Jan-Erik Tesdal[1,*], **Ryan Abernathey**[1] **and Ian Fenty**[2]

[1] Lamont-Doherty Earth Observatory, Columbia University, Palisades, NY, USA
[2] Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, USA
[*] *Corresponding author*: jt2796@columbia.edu

This section demonstrates the closure of the global heat budget in ECCOv4. The steps and Python code has been directly derived from the calculations and MATLAB code in *"A Note on Practical Evaluation of Budgets in ECCO Version 4 Release 3"* by Christopher G. Piecuch (https://ecco.jpl.nasa.gov/drive/files/Version4/Release3/doc/v4r3_budgets_howto.pdf).

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Ryan Abernathey Jan-Erik Tesdal. Lamont-doherty earth observatory, columbia university, palisades, ny, usa. 2020.

[2] Ian Fenty. Jet propulsion laboratory, california institute of technology, pasadena, ca, usa. 2020.

[3] Alistair Adcroft and Jean-Michel Campin. Rescaled height coordinates for accurate representation of free-surface flows in ocean circulation models. *Ocean Modelling*, 7(3):269 – 284, 2004. URL: http://www.sciencedirect.com/science/article/pii/S1463500303000544, doi:https://doi.org/10.1016/j.ocemod.2003.09.003.