Rosely Documentation

John Volk

Mar 11, 2021

Contents

Interactive wind rose diagrams made easy using `plotly` and `pandas`

View on GitHub

CHAPTER 1

Installation

`Rosely`'s dependencies are Python 3.4+, NumPy, pandas, and Plotly.

You may install the dependencies using the conda virtual environment (recommended), the environment file can be downloaded here and installed and activated by

```
conda env create -f environment.yml
conda activate rosely
```

Once activated install with PIP:

```
pip install rosely
```

If all went well you should be able to import

```
>>> from rosely import WindRose
```

CHAPTER 2

Tutorial

This tutorial covers basic usage of the `Rosely` package including loading of data, calculatiion of wind statistics, and wind rose plotting customizations.

```
>>> import pandas as pd
>>> import plotly.express as px
>>> from rosely import WindRose
```

2.1 Read input data

rosely requires wind data to first be loaded into a `pandas.DataFrame` object, also wind direction should be in degrees, i.e. in [0, 360].

The example data used in this tutorial is a modified version of 30 minute data that was originally from the "Twitchell Alfalfa" AmeriFlux eddy covariance flux tower site in the Sacramento–San Joaquin River Delta in California. The site is located in alfalfa fields and exhibits a mild Mediterranean climate with dry and hot summers, for more information on this site click here.

The data used for this example can be downloaded on the Rosely GitHub repositor here. And a Jupyter Notebook of this tutorial is available here.

```
>>> df = pd.read_csv('test_data.csv', index_col='date', parse_dates=True)
>>> df[['ws','wd']].head()
```

Or another view of the summary statistics of wind data

```
>>> df[['ws','wd']].describe()
```

2.2 Create a `WindRose` instance

Using the loaded wind speed and direction data within a `pandas.DataFrame` we can initialize a *`rosely.WindRose`* object which provides simple methods for generating interactive wind rose diagrams.

```
>>> WR = WindRose(df)
```

Alternatively the dataframe can be later assigned to a *`WindRose`* object,

```
>>> WR = WindRose()
>>> WR.df = df
```

2.3 Calculate wind statistics

A wind rose diagram is essentially a stacked histogram that is binned by wind speed and freqeuncy for a set of wind directions. These calculations are accomplished by the *`WindRose.calc_stats()`* method which allows for changing the number of default wind speed bins (equally spaced) and whether or not the frequency is normalized to sum to 100 or it is just the actual frequency of wind occurences (counts) in a certain direction and speed bin.

By default the freqeuncy is normalized and the number of wind speed bins is 9:

```
>>> WR.calc_stats()
```

To view the results of the wind statistics that will be used for the wind rose later, view the `WindRose.wind_df` which is created after running *`WindRose.calc_stats()`*:

```
>>> # view all statistics for winds coming from the North
>>> WR.wind_df.loc[WR.wind_df.direction=='N']
```

Note: The winds speed bins in a certain direction may appear to be duplicated above but they are not, what is happening is that *`WindRose.calc_stats()`* bins each direction on a 16 point compass twice for 11.25 degrees sections on both sides of the compass azimuth. So for North there are two internal azimuth bins: from 348.75-360 degrees and from 0-11.25 degrees. If you wanted to see the summed Northerly winds frequencies within the 9 speed bins you could run:

```
>>> WR.wind_df.groupby(['direction','speed']).sum().loc['N']
```

Here is an example of not normalizing the freqeuncy (using raw counts instead) and using 6 instead of 9 bins for speed. This example shows the same grouped output for Northerly winds,

```
>>> WR.calc_stats(normed=False, bins=6)
>>> WR.wind_df.groupby(['direction','speed']).sum().loc['N']
```

Lastly, if the wind speed and wind direction columns in the dataframe assigned to the *`WindRose`* object are not named 'ws' and 'wd' respectively, instead of renaming them ahead of time or inplace, you may pass a dictionary that maps their names to the *`WindRose.calc_stats()`* method. For example, lets purposely change the names in our input dataframe to 'wind_speed' and 'direction':

```
>>> tmp_df = df[['ws','wd']]
>>> tmp_df.columns = ['wind_speed', 'direction']
>>> tmp_df.head()
```

Now reassign this differently named dataframe to a *WindRose* instance to demonstrate

```
>>> WR.df = tmp_df
>>> # create renaming dictionary
>>> names = {
>>>     'wind_speed':'ws',
>>>     'direction': 'wd'
>>> }
>>> WR.calc_stats(normed=False, bins=6, variable_names=names)
>>> WR.wind_df.groupby(['direction','speed']).sum().loc['N']
```

The same results were achieved as above, however the column names used for initial assignment are retained by the *WindRose.df* property:

```
>>> WR.df.head()
```

Tip: In this tutorial the full dataset of 30 minute windspeed was used to create the statistics (above) and the diagrams (below), in practice it may be important to view wind speed / direction during certain time periods like day or night, or summer/winter seasons. This is one of the main reasons for using `pandas.DataFrame` objects- they have many tools for time series analysis, particularly temporal aggregation and resampling. If you wanted to view the wind statistics/plot for this site during day times defined (not quite accurately) as 8:00 AM to 8:00 PM it is as simple as this:

```
>>> # reassign the wind data but sliced just for day hours we want
>>> WR.df = df[['ws','wd']].between_time('8:00', '16:00')
>>> # calculate the wind statistics again
>>> WR.calc_stats(normed=False, bins=6)
>>> WR.wind_df.groupby(['direction','speed']).sum().loc['N']
```

2.4 Generate wind rose diagrams

The main purpose of *rosely* is to simplyfy the generation of beautiful, interactive wind rose diagrams by using `plotly.express.bar_polar` charts and `pandas`. Once a *WindRose* object has been created and has been assigned a `pandas.DataFrame` with wind speed and wind direction you can skip calculating statistics (falls back on default parameters for statistics) and jump right to creating a wind rose diagram. For example:

```
>>> # create a new WindRose object from our example data with 'ws' and 'wd' columns
>>> WR = WindRose(df)
>>> WR.plot()
    Wind speed and direction statistics have not been calculated, Calculating them␣
↪now using default parameters.
```

The two lines above saved the plot with default parameters (9 speed bins) normalized frequency, and default *rosely* color schemes to the current working directory named 'windrose.html'.

To view the default plot without saving,

```
>>> # try zooming, clicking on legend, etc.
>>> WR.plot(output_type='show')
```

Notice that these plots used the default statistics parameters, to use other options be sure to call *WindRose.calc_stats()* before *WindRose.plot()*. E.g. if we wanted 6 equally spaced bins with freqeuncies represented as counts as opposed to percentages,

```
>>> WR.calc_stats(normed=False, bins=6)
>>> WR.plot(output_type='show')
```

Hint: Assign the path to save the output file if `output_type` = 'save' using the `out_file` keyword argument.

The third option that can be assigned to `output_type` other than 'save' and 'show' is 'return'. When `output_type='return` the *WindRose.plot()* method returns the plot figure for further modification or integration in other workflows like adding it into a group of subplots.

Here is an example use of the 'return' option that modifies the wind rose after it's creation by *rosely* by changing the background color and margins:

2.5 Easy wind rose customizations

rosely makes it simple to experiment with different wind rose statistcs options but also plot color schemes, this section of the tutorial highlights some useful options to the *WindRose.plot()* method for doing the latter.

First off there are three important keyword arguments to *WindRose.plot()* that control the color schemes (`colors`, `template`, and `colors_reversed`):

1. `colors` is the name of the `Plotly` sequential color swatch or a list of your own RGB or Hex colors to passfor the stacked histograms (the first color in the list will be the most inner color on the diagram and them moving outwards towards higher wind speeds).
2. `template`, this is the name of the `Plotly` template that defines the background color and other visual appearences. You may also pass a custom `Plotly.py` template object.
3. `colors_reversed` simply allows for the automatic reversal of color sequences which may be useful because some color swatches range from light to dark while others range from dark to light tones.

A list of all provided colors (hint hover over them to view the Hex or RGB values themselves):

As for templates they are easily listed by the following:

```
>>> import plotly.io as pio
>>> pio.templates
    Templates configuration
    -----------------------
        Default template: 'plotly'
        Available templates:
            ['ggplot2', 'seaborn', 'plotly', 'plotly_white', 'plotly_dark',
             'presentation', 'xgridoff', 'none']
```

Now, let's try out some of these colors and templates!

```
>>> WR.plot(output_type='show', template='seaborn', colors='Plotly3', width=600,␣
↪height=600)
```

Some color swatches may look better without colors reversed,

```
>>> WR.plot(output_type='show', template='xgridoff', colors='turbid', colors_
↪reversed=False)
```

This final example not only shows different color schemes but that you can pass additional useful keyword arguments that are accepted by `plotly.express.bar_polar` such as `title`, and `width` to *WindRose.plot()*. It

also demonstrates that HTML can be embedded into the plot title and an example of prefiltering the wind time series to before calculating wind statistics, in this case to create a wind rose for the winter months only.

```
>>> # reassign the wind data but sliced just for Dec-Mar
>>> WR.df = df[['ws','wd']].loc[df.index.month.isin([12,1,2,3])]
>>> # calculate the wind statistics (only necessary because not using default n bins)
>>> WR.calc_stats(normed=True, bins=6)
>>> WR.plot(
>>>     output_type='show',
>>>     colors='Greens',
>>>     template='plotly_dark',
>>>     colors_reversed=False,
>>>     width=600,
>>>     height=600,
>>>     title='Eddy Flux Site on Twitchell Island, CA <br>Wind measured Dec-Mar<br><a␣
↪href="https://ameriflux.lbl.gov/sites/siteinfo/US-Tw3">Visit site</a>'
>>> )
```

As we can see the winter wind system is substantially different from the average long-term wind which may be expected due to seasonal storm systems or temporally varying larger scale atmospheric circulations.

CHAPTER 3

API Reference

This page documents objects and methods provided by `Rosely`.

3.1 Module contents

A small package for efficiently generating customizable and interactive wind rose diagrams. Once wind speed and direction is loaded into a `pandas.DataFrame` the package can create wind speed and direction statistics which are used to create windrose diagrams via `Plotly`'s polar bar chart function with multiple tools for easy plot customization.

3.2 WindRose class

class rosely.**WindRose**(*df=None*)

Bases: `object`

Manage data for calculating wind statistics and provide simple interface for creating customizable wind rose diagrams.

df

arbitrary `pandas.DataFrame` that is assigned to a *`WindRose`* object that must contain wind speed and direction columns before using other *`WindRose`* methods.

Type `pandas.DataFrame`

theta_labels

16 point compass labels for wind rose diagrams.

Type list

theta_angles

array of 11.25 degree intervals for 16 point compass.

Type `numpy.ndarray`

wind_df
calculated wind statistics produced by *WindRose.calc_stats()* and used by *WindRose.plot()*.

Type pandas.DataFrame

calc_stats(*normed=True*, *bins=9*, *variable_names=None*)
Calculate wind speed and direction bins needed for generating wind rose diagrams.

After running *WindRose.calc_stats()* with different options a new instance attribute *WindRose.wind_df* is generated that contains the binned wind speed statistics. This attribute is in the form of a pandas.DataFrame and can be used to create a histogram or saved to disk.

Keyword Arguments

- **normed** (*bool*) – default True. If True compute wind speed/direction frequency bins that are normalized to sum to 100. If False frequency bins are counts of occurences of wind speed/direction.
- **bins** (*int or list*) – default 9. Number of wind speed and direction bins to calculate. 9 is used because most plotly color sequences are lenght 9 or 10 which are later used by *WindRose.plot()*
- **variable_names** (*None or dict*) – default None. If none the wind speed and wind direction columns in *WindRose.df* should be named 'ws' and 'wd' respectively. Otherwise a dictionary that maps the respective columns to 'ws' and 'wd' should be provided.

Returns None

Example

Assuming you have a pandas.DataFrame loaded that has wind speed and direction columns titled 'wind_speed' and 'wind_direction' and the dataframe is named df:

```
>>> from rosely import WindRose
>>> WR = WindRose(df)
>>> names = {'wind_speed':'ws', 'wind_direction':'wd'}
>>> WR.calc_stats(normed=False, bins=8, variable_names=names)
```

Now WR.wind_df should have the appropirate statistics and the *WindRose.plot()* will use these statistics for the polar stacked histogram (wind rose).

df
pandas.DataFrame containing input time series wind data needed to run *WindRose.plot()*.

plot(*output_type='save'*, *out_file=None*, *colors='Plasma'*, *template='plotly_dark'*, *colors_reversed=True*, ***kwargs*)
Create interactive wind rose diagrams with easily customizable options using Plotly's polar bar chart.

Keyword Arguments

- **output_type** (*str*) – default 'save'. If 'save' save graph to out_file. Other options: 'show' will show in a new tab in web browser or within a Jupyter Notebook, and 'return' will return the plotly figure for further manual customization/modification or use in custom workflows like saving as a subplot with other plot figures.
- **out_file** (*None or str*) – default None. If output_type='save' then save to specified path, if None save to current working directory as "windrose.html".
- **colors** (*str*) – default 'Plasma'. Name of Plotly color swatch or sequence to use for coloring bins from center outward on wind rose. See *Tutorial* for examples and all options. Can also pass a list of hex or rgb colors of your own.

- **template** (*str*) – default 'plotly_dark'. Name of Plotly template for background theme/colors on wind rose.
- **colors_reversed** (*bool*) – True. If True reverse the colors in `colors`. The first color in the sequence will be used for the lightest wind speed bin.
- ****kwargs** – other keyword arguments are passed to the `plotly.express.bar_polar` plot function, e.g. title or width.

Returns (None or `plotly.graph_objects.Figure`)

Example

Assuming a `pandas.DataFrame` object called "df" has been loaded and contains columns 'ws' and 'wd' with wind speed and direction,

```
>>> from rosely import WindRose
>>> WR = WindRose(df) # df already loaded pandas dataframe
>>> # if you skip running WR.calc_stats the defaults will be used
>>> WR.plot(output_type='show', colors='Greens',
>>>     colors_reversed=False)
```

This will produced a normalized frequency wind rose (frequency 0-100 percent) with 9 bins. To specify the to use count frequency or a different number of bins use the *WindRose.calc_stats()* method before running *WindRose.plot()*.

Tip: To see a list of all provided color sequences provided by Plotly,

```
>>> import plotly.express as px
>>> px.colors.sequential.swatches()
```

All of the listed color schemes can be passed to the `colors` arugment of *WindRose.plot()*.

---

theta_angles
Used by autodoc_mock_imports.

theta_labels = ['N', 'NNE', 'NNE', 'NE', 'NE', 'ENE', 'ENE', 'E', 'E', 'ESE', 'ESE', '

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index