Process: Creating Wind Rose Plots with Python

Jashan Chopra
06/14/2019

I. Introduction

At Summit, Greenland, air can be polluted by the power generating processes at the camp. Since meteorological data is measured at Summit, we can identify the wind direction at the time of our sampled atmospheric concentrations, and use this to filter out undesirable data. In this scenario, the Summit clean air management plan identifies air at TAWO to be impacted when the wind direction is between 342 degrees and 72 degrees, or when the wind speed is less than 2 knots. The ideal way to visualize this process is with the Wind Rose Plot, and this guide will go over the steps to creating these plots in entirety, using Python 3+.

II. Data Wrangling

Acquiring Met Data

In order to match atmospheric concentration values with their direction, meteorological data with similar timestamps to the data must be acquired. Although the source of this data may depend on the location, the NOAA ESRL GMD is a good starting point. Met data by location can be found at ftp://ftp.cmdl.noaa.gov/met.

Data from this source is broken into files by year, the following code block provides a simple method for reading in the data for this specific case. Additional data trimming can be performed, such as the removal of NaN values with *df.dropna()*. Python datetime is a universal datetime format, and supports a wide variety of excellent features. An additional step in this script is to create a datetime column from the existing date columns that we have. The libraries imported in this step will cover the entire procedure.

```python
# import libraries and functions
import matplotlib.pyplot as plt
from WindRose.metTrim import metTrim, createDatetime
import pandas as pd
import matplotlib.cm as cm
import windrose


def createDatetime(yr, mo, dy, hr):
    Simple function to convert four numpy arrays of seperate date values into a single
        datetime array

    datetime = []
    for i in range(len(yr)):
        time = dt.datetime(yr[i], mo[i], dy[i], hr[i])
        datetime.append(time)

    return datetime


# ——— initial reading of data
root = r'C:\Users\ARL\Desktop\MetData'
ext = list(range(12, 20))
```

```

colnames = ['na', 'yr', 'mo', 'dy', 'hr', 'dir', 'spd', 'steady', 'na', 'na', 'na', 'na', '
    na', 'na']
met = pd.DataFrame(columns=colnames)
for yr in ext:
    # read in data
    data = pd.read_csv(root + r'\met_sum_insitu_1_obop_hour_20{}.txt'.format(yr),
        delim_whitespace=True,
                       header=None)
    data.columns = colnames
    met = met.append(data)
print('Data Imported')

# ---- trimming data
met = met.drop('na', axis=1)
met = met.replace(-999.9, np.nan)
met = met.replace(-9, np.nan)
met = met.replace(-999, np.nan)
met = met.replace(-99.9, np.nan)
met = met.dropna(axis=0, how='any')

# ---- convert date to datetime
metInt = met.applymap(int)
dates = createDatetime(metInt['yr'].values,
                       metInt['mo'].values,
                       metInt['dy'].values,
                       metInt['hr'].values)

met['datetime'] = dates                                                     # add it
     as a new column
met = met.drop(['yr', 'mo', 'dy', 'hr'], axis=1)                            # drop
    old date columns
```

Combining Met Data with Atmospheric Data

The central difficulty with this task is combining the meteorological data with your data. With multiple datasets, the easiest way to do this is also with a function. First we import our met data that we just made, and then read our csv of atmospheric data. Similar to with the met data, we create a datetime column if one does not already exist. Initial trims on the data include dropping early values from met data, and removing a column for speed. The final step uses an incredible built in function in Pandas called *pd.merge_asof*. This function takes the second specified data set (the met data) and merges it onto the first one (our data) with conditions. First, we perform the action on the datetime column that we made before, we find the nearest direction value, and we find it within the hour, although a minute tolerance can be set as well.

```
# import libraries and functions
import matplotlib.pyplot as plt
from WindRose.metTrim import metTrim, createDatetime
import pandas as pd
import matplotlib.cm as cm
import windrose


def metCombo(filename):

    # —— import data
    met = metTrim()
    sheet = pd.read_csv(filename, encoding='utf8', delim_whitespace=True)

    # —— data organization
    colnames = ['yr', 'mo', 'day', 'hr', 'min', 'val']
        # column names
    sheet.columns = colnames
        # rename cols
    sheet['datetime'] = createDatetime(sheet['yr'],
        # create datetime
                                       sheet['mo'],
                                       sheet['day'],
                                       sheet['hr'])
    sheet.drop(['yr', 'mo', 'day', 'hr', 'min'], axis=1, inplace=True)
        # drop old cols

    # —— trimming data
    earlyVals = (met['datetime'] <= sheet['datetime'][0])                       # early met
        vals
    met.drop(earlyVals, axis=0, inplace=True)                                   # trim early
         vals
    met.reset_index(drop=True, inplace=True)                                    # reset
        index
    met.drop(['steady'], axis=1, inplace=True)                                  # remove
        some columns

    # merge the met data onto the concentration data by finding the nearest datetime within
        an hour
    combo = pd.merge_asof(sheet, met, on='datetime',
                          direction='nearest',
                          tolerance=pd.Timedelta('1 hour'))

    return combo
```

III. Plotting

Finally we have to create the plots, which require the custom windrose python library. It can be installed from the Pip index with **pip install windrose**. These graphs usually look best stand alone, or with two subplots. Increased plots will lose visibility and also have annoying legends. First we set the filepaths to our data, and then call our previous functions. We then setup the subplots calling the windrose projection, and set a central title for the subplots. We then call the matplotlib bar plot function. There are multiple inputs here that required some exploration.

- *normed* : Boolean, determines if the ring numbers are normalized or if they represent the actual amount of representative data points.
- *opening* : Determines the spacing between bars, 1 is the maximum and means no spacing. 0.9 looks nice.
- *edgecolor* : Determines the border color of the bars, white is the default and will look as if there was no border.
- *nsector* : The number of actual bars on the plot, this number depends on how precise you need to identify regions where the wind is coming from.
- *bins* : Similarly, this is the number of bins. Depends on the range between high and low values in the actual dataset, as with any other bar graph.
- *cmap* : This takes a matplotlib colormap as an input, and changes how the sectors and bins are colored.
- *blowto* : This last one is important to consider. By default, the graph shows directions that the **wind is coming**

from, setting this variable to True will reverse this and show the direction that the wind is blowing to.

```python
def windRose():
    root = r'C:\Users\ARL\Desktop\J_Summit\analyses\HarmonicFit\textfiles'          # root
        source
    ethPath = root + r'\ethane.txt'
    acePath = root + r'\acetylene.txt'

    ethane = metCombo(ethPath)
    ace = metCombo(acePath)

    # ---- plotting
    # setup subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, subplot_kw=dict(projection='windrose'))
                                    #
    fig.suptitle('NMHC Conc. Residuals at Summit by Wind Direction', fontsize=16)
    plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=0.2, hspace
        =-0.2)

    # setup ethane windrose
    ax1.bar(ethane['dir'].values, ethane['val'].values, normed=False, opening=0.9, edgecolor
        ='black',
            nsector=24, bins=14, cmap=cm.viridis_r, blowto=False)
    ax1.set_title('Summit Ethane Conc. Residual [ppb]\n')
    ax1.set_legend(loc=8, fancybox=True, shadow=True, bbox_to_anchor=(0.70, -.45))

    # setup acetylene windrose
    ax2.bar(ace['dir'].values, ace['val'].values, normed=False, opening=0.9, edgecolor='
        black',
            nsector=24, bins=6, cmap=cm.viridis_r, blowto=False)
    ax2.set_title('Summit Ace Conc. Residual [ppb]\n')
    ax2.set_legend(loc=8, fancybox=True, shadow=True, bbox_to_anchor=(0.5, -.35))

    plt.show()
```

All that's left is to set the title and the legend. There are a few options that make the legend nicer, setting fancybox and shadow to true will give a better look. The "bbox_to_anchor" feature provides the ability to move the legend around, but requires some messing around with.

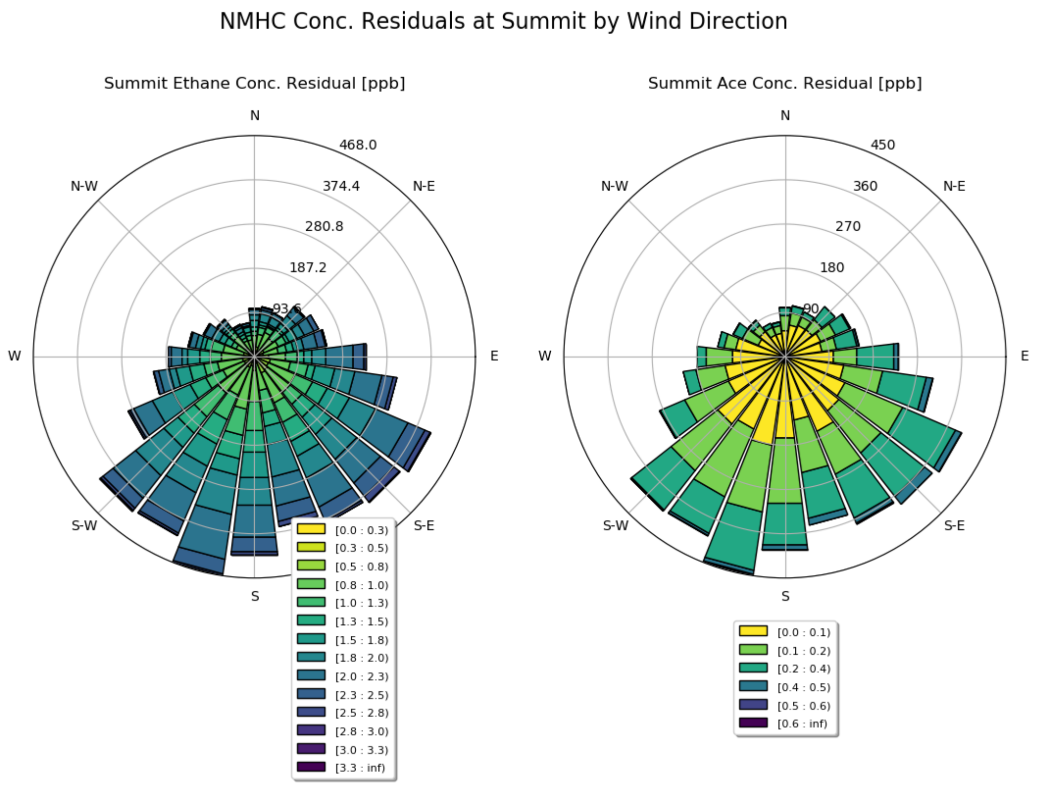

Fig. 1 Another Example with NMHC data

When combined with a map to identify potential pollution sources, the windrose map can be extremely useful. If you have a set of identified directions to cut from the data, it will be simple with indexing the dataframes you have already developed. This concludes the windrose plotting documentation.