python4oceanographers (https://ocefpaf.github.io/python4oceanographers/)

Turning ripples into waves

- Atom (http://ocefpaf.github.io/python4oceanographers//atom.xml)

Search

» Atom

- Blog (http://ocefpaf.github.io/python4oceanographers)
- Archives (http://ocefpaf.github.io/python4oceanographers/archives.html)
- About (http://ocefpaf.github.io/homepage)

Exploring GPX files

Aug 18, 2014

This post is an exercise on how to explore GPX files. Most of what I did in this post learned from this tutorial (http://nbviewer.ipython.org/github/titsworth/hsvpy-runtalk/tree/master/).

Deep down the GPX file format is just a XML document text. They can be parsed with any XML parser out there, but the gpxpy (http://www.trackprofiler.com/gpxpy/index.html) module makes that task much easier. Here is a quick example on how to load and explore the data inside a GPX file.

In [2]:
```
import gpxpy
gpx = gpxpy.parse(open('./data/2014_08_05_farol.gpx'))

print("{} track(s)".format(len(gpx.tracks)))
track = gpx.tracks[0]

print("{} segment(s)".format(len(track.segments)))
segment = track.segments[0]

print("{} point(s)".format(len(segment.points)))
```

```
1 track(s)
1 segment(s)
1027 point(s)
```

Now let's extract the data for all those points. Here I have 1 track and 1 segment, but a GPX file might contain multiple tracks and segments. The best practice here is to always loop through all tracks and segments.

In [3]:
```
data = []
segment_length = segment.length_3d()
for point_idx, point in enumerate(segment.points):
    data.append([point.longitude, point.latitude,
                 point.elevation, point.time, segment.get_speed(point_idx)])

from pandas import DataFrame

columns = ['Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
df = DataFrame(data, columns=columns)
df.head()
```

Out[3]:

	Longitude	Latitude	Altitude	Time	Speed
0	-38.502595	-13.005390	10.9	2014-08-05 17:52:49.330	NaN
1	-38.502605	-13.005415	11.8	2014-08-05 17:52:49.770	2.672951
2	-38.502575	-13.005507	11.7	2014-08-05 17:52:54.730	3.059732
3	-38.502545	-13.005595	11.6	2014-08-05 17:52:57.750	4.220779
4	-38.502515	-13.005680	11.4	2014-08-05 17:53:00.720	3.939967

I want to plot the direction of the movement with a quiver plot. For that I will need the `u` and `v` velocity components. And to compute `u` and `v` I need the angle associated to each speed data. Instead of re-inventing the wheel I will use the `seawater` library `sw.dist` function to calculate the angles.

I also smoothed the data a little bit to improve the plot. (GPX data from smart-phones can be very noisy.)

In [4]:
```
import numpy as np
import seawater as sw
from oceans.ff_tools import smoo1

_, angles = sw.dist(df['Latitude'], df['Longitude'])
angles = np.r_[0, np.deg2rad(angles)]

# Normalize the speed to use as the length of the arrows
r = df['Speed'] / df['Speed'].max()
kw = dict(window_len=31, window='hanning')
df['u'] = smoo1(r * np.cos(angles), **kw)
df['v'] = smoo1(r * np.sin(angles), **kw)
```

Now let's use mplleaflet (https://github.com/jwass/mplleaflet) to plot the track and the direction.

In [5]:
```
import mplleaflet
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
df = df.dropna()
ax.plot(df['Longitude'], df['Latitude'],
        color='darkorange', linewidth=5, alpha=0.5)
sub = 10
ax.quiver(df['Longitude'][::sub], df['Latitude'][::sub], df['u'][::sub], df['v'][::sub], col
pskyblue', alpha=0.8, scale=10)
mplleaflet.display(fig=fig, tiles='esri_aerial')
```

Out[5]:

Leaflet | mplleaflet | Tiles © Esri — Source: Esri, i-cubed, USDA, USGS, AEX, GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community

If you have tons of GPX files with your run data, it might come in handy to define a function to read them all at once.

In [6]:
```
import os
from glob import glob

def load_run_data(gpx_path, filter=""):
    gpx_files = glob(os.path.join(gpx_path, filter + "*.gpx"))
    run_data = []
    for file_idx, gpx_file in enumerate(gpx_files):
        gpx = gpxpy.parse(open(gpx_file, 'r'))
        # Loop through tracks
        for track_idx, track in enumerate(gpx.tracks):
            track_name = track.name
            track_time = track.get_time_bounds().start_time
            track_length = track.length_3d()
            track_duration = track.get_duration()
            track_speed = track.get_moving_data().max_speed

            for seg_idx, segment in enumerate(track.segments):
                segment_length = segment.length_3d()
                for point_idx, point in enumerate(segment.points):
                    run_data.append([file_idx, os.path.basename(gpx_file), track_idx, track_
                                     track_time, track_length, track_duration, track_speed,
                                     seg_idx, segment_length, point.time, point.latitude,
                                     point.longitude, point.elevation, segment.get_speed(poi
])
    return run_data
```

In [7]:
```
data = load_run_data(gpx_path='./data/GPX/', filter="")
df = DataFrame(data, columns=['File_Index', 'File_Name', 'Index', 'Name',
                              'Time', 'Length', 'Duration', 'Max_Speed',
                              'Segment_Index', 'Segment_Length', 'Point_Time', 'Point_Latitu
                              'Point_Longitude', 'Point_Elevation', 'Point_Speed'])

HTML(df.head().to_html(max_cols=4))
```

Out[7]:

	File_Index	File_Name	...	Point_Elevation	Point_Speed
0	0	2013_08_04_USP.gpx	...	764.0	NaN
1	0	2013_08_04_USP.gpx	...	767.0	1.726115
2	0	2013_08_04_USP.gpx	...	769.5	3.601075
3	0	2013_08_04_USP.gpx	...	772.0	3.540769
4	0	2013_08_04_USP.gpx	...	774.5	3.025701

Here I will clean up the DataFrame and convert the distances to km.

In [8]:
```
cols = ['File_Index', 'Time', 'Length', 'Duration', 'Max_Speed']
tracks = df[cols].copy()
tracks['Length'] /= 1e3
tracks.drop_duplicates(inplace=True)
tracks.head()
```

Out[8]:

	File_Index	Time	Length	Duration	Max_Speed
0	0	2013-08-04 16:10:00	7.562737	6481	4.473565
712	1	2013-04-15 18:05:00	6.504129	2455	6.344318
877	2	2013-08-08 17:18:00	7.746483	2620	5.609248
1477	3	2013-04-22 17:42:00	7.281408	2445	5.618331
2192	4	2013-09-05 16:36:00	7.628724	4540	3.321567

And finally let's add a Track Year and Month columns based on track time. That way we can explore the run data with some stats and bar plots.

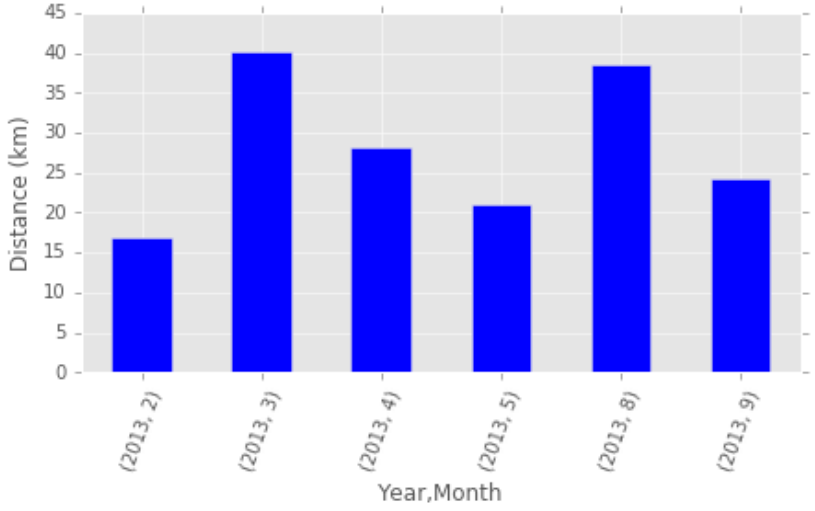
In [9]:
```
tracks['Year'] = tracks['Time'].apply(lambda x: x.year)
tracks['Month'] = tracks['Time'].apply(lambda x: x.month)
tracks_grouped = tracks.groupby(['Year','Month'])
tracks_grouped.describe().head()
```

Out[9]:

			Duration	File_Index	Length	Max_Speed
Year	Month					
2013	2	count	2.000000	2.000000	2.000000	2.000000
		mean	2924.000000	12.000000	8.419711	4.691473
		std	124.450793	8.485281	0.024573	0.740730
		min	2836.000000	6.000000	8.402335	4.167698
		25%	2880.000000	9.000000	8.411023	4.429585

In [10]:
```
figsize=(7, 3.5)

tracks_grouped = tracks.groupby(['Year', 'Month'])
ax = tracks_grouped['Length'].sum().plot(kind='bar', figsize=figsize)
xlabels = [text.get_text() for text in ax.get_xticklabels()]
ax.set_xticklabels(xlabels, rotation=70)
_ = ax.set_ylabel('Distance (km)')
```



Bad news! My goal was to run 50 km per month... I am clear way too far from accomplishing it! (Not to mentioned the fact that there is no data from 2014!)

To close this post I want to produce a plot similar to this (http://flowingdata.com/2014/02/05/where-people-run/) using my run data.

In [11]:
```
def load_run_data(gpx_path, filter=""):
    gpx_files = glob(os.path.join(gpx_path, filter+"*.gpx"))
    run_data = []
    for file_idx, gpx_file in enumerate(gpx_files):
        try:
            gpx = gpxpy.parse(open(gpx_file, 'r'))
        except:
            os.remove(gpx_file)
            continue
        run_data_tmp = [[file_idx, gpx_file, point.latitude,point.longitude, point.elevation
                         for track in gpx.tracks
                             for segment in track.segments
                                 for point in segment.points]
        run_data += run_data_tmp
    return run_data

def clear_frame(ax):
    ax.xaxis.set_visible(False)
    ax.yaxis.set_visible(False)
    for spine in ax.spines.values():
        spine.set_visible(False)

def plot_run_data(coords, **kwargs):
    columns = ['Index', 'File_Name', 'Latitude', 'Longitude', 'Altitude']
    coords_df = DataFrame(coords, columns=columns)
    grouped = coords_df.groupby('Index')
    fig, ax = plt.subplots(figsize=kwargs.get('figsize', (13 ,8)))

    bgcolor = kwargs.get('bgcolor', '#001933')
    color = kwargs.get('color', '#FFFFFF')
    linewidth = kwargs.get('linewidth', .035)
    alpha = kwargs.get('alpha', 0.5)

    kw = dict(color=color, linewidth=linewidth, alpha=alpha)
    for k, group in grouped:
        ax.plot(group['Longitude'], group['Latitude'], **kw)
    ax.grid(False)
    ax.patch.set_facecolor(bgcolor)
    ax.set_aspect('auto','box','C')
    clear_frame(ax)
    fig.subplots_adjust(left=0, right=1, top=1, bottom=.1)
    return ax
```

In [12]:
```
df = load_run_data(gpx_path='./data/GPX/')
ax = plot_run_data(df, figsize=(4, 3), alpha=0.85,
                   bgcolor='#0A2A0A')
_ = ax.axis([-46.74, -46.71, -23.57, -23.55])
```

I tried to find public run data for Salvador to discover the best places to run here using that kind of plot. First I tried RunKeeper (http://runkeeper.com/), the app does make their public data available online, but it is not a popular app in Brazil and I could not find any tracks for Salvador in the database. Sportstraker (http://www.sports-tracker.com/), on the other hand, is very popular here. But Sportstraker do not publish the public data online.

If you read this and have some GPX files data from your training and want to see a map of Salvador most popular places to run, get in touch!

In [13]:
```
HTML(html)
```

Out[13]: This post was written as an IPython notebook. It is available for download (https://ocefpaf.github.com/python4oceanographers/downloads/notebooks/2014-08-18-gpx.ipynb) or as a static html (https://nbviewer.ipython.org/url/ocefpaf.github.com/python4oceanographers/downloads/notebooks/2014-08-18-gpx.ipynb).

Posted by Filipe Fernandes Aug 18, 2014

Comments

Recent Posts

- Plotting a GeoDataFrame with folium (https://ocefpaf.github.io/python4oceanographers/blog/2015/12/14/geopandas_folium/)
- Python tools for sgrid (https://ocefpaf.github.io/python4oceanographers/blog/2015/12/07/pysgrid/)
- Active Fire Data WMS Modis layer (https://ocefpaf.github.io/python4oceanographers/blog/2015/11/30/chapada/)
- Desabafo de um Pesquisador (https://ocefpaf.github.io/python4oceanographers/blog/2015/11/23/FAP/)
- Displaying a field of arrows with folium (https://ocefpaf.github.io/python4oceanographers/blog/2015/11/16/folium_quiver/)

Blogroll

- PyAOS (http://pyaos.johnny-lin.com/)
- EarthPy (http://earthpy.org/)
- PyHOGs (http://pyhogs.github.io/)
- drclimate (http://drclimate.wordpress.com/)
- Software Carpentry (http://software-carpentry.org/blog/index.html)