kaggle Search Competitions Datasets Kernels Discussion Learn Sign In

How to: Folium for maps, heatmaps & time analysis

Python notebook using data from 1.6 million UK traffic accidents · 2,577 views

6 Fork 3

Version 2
3 commits

forked from How to: Folium for maps, heatmaps & time analysis

Notebook
Data
Log
Comments

Notebook Data Log Comments

What is Folium

Folium is a tool that makes you look like a mapping God while all the work is done in the back end.

It's a Python wrapper for a tool called leaflet.js. We basically give it minimal instructions, JS does loads of work in the background and we get some very, very cool maps. It's great stuff.

For clarity, the map is technically called a 'Leaflet Map'. The tool that let's you call them in Python is called 'Folium'.

The other cool stuff?

It gives you interactive functionality. Want to let users drop markers on the map? Can do. Build heatmaps? Can do. Build heatmaps that change with time? Can do.

Funk yeah! Let's do this.

In [1]:

```
# This Python 3 environment comes with many helpful an
alytics libraries installed
# It is defined by the kaggle/python docker image: htt
ps://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
 in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O
 (e.g. pd.read_csv)
import folium

#df_traffic = pd.read_csv('../input/ukTrafficAADF.cs
v')
df_acc = pd.read_csv('../input/accidents_2005_to_200
7.csv', dtype=object)
```

In [2]:

```
map_hooray = folium.Map(location=[51.5074, 0.1278],
                    zoom_start = 11) # Uses lat then l
on. The bigger the zoom number, the closer in you get
map_hooray # Calls the map to display

#map_osm = folium.Map(location=[54.7, -4.36])
#This 2nd set of coordinates will drop you down right
 in the middle of the UK, which is actually the seas b
```

```
ecause it's between mainland and N.Ireland,
```

Out[2]:

Fun visual styles

In [3]:

```
t_list = ["Stamen Terrain", "Stamen Toner", "Mapbox B
right"]
map_hooray = folium.Map(location=[51.5074, 0.1278],
                        tiles = "Stamen Terrain",
                        zoom_start = 12)
map_hooray
```

Out[3]:

In [4]:

```
map_hooray = folium.Map(location=[51.5074, 0.1278],
                        tiles = "Stamen Toner",
                        zoom_start = 12)
```

```
map_hooray
```

Out[4]:

Markers

These are defined outside the map. This is similar to a basmap. Once you've set the location, zoom, style, i.e. the place, everything else is an addition that's placed over the top, so it's called and added to (.add_to) the map.

Note the 'popup attribute. This text appears on clicking the map.

In [5]:

```
map_hooray = folium.Map(location=[51.5074, 0.1278],
                        tiles = "Stamen Toner",
                        zoom_start = 12)
# 'width=int' and 'height=int' can also be added to th
e map

folium.Marker([51.5079, 0.0877], popup='London Bridg
e').add_to(map_hooray)
map_hooray
```

Out[5]:

Leaflet (http://leafletjs.com)

More complex markers

The London Bridge marker is the same as above, I just added a little colour with that extra line of green

The second marker is more interesting. First, it's not a marker like the pin. The pin is an icon type marker. The circle is essentially just a coloured overlay, so we use a different colour command.

CircleMarker radius is set in pixels so if you change the zoom you need to change the pixels. It can also take a fill_color that's semi-transparent.

Interactive markers

Let your users add marker with

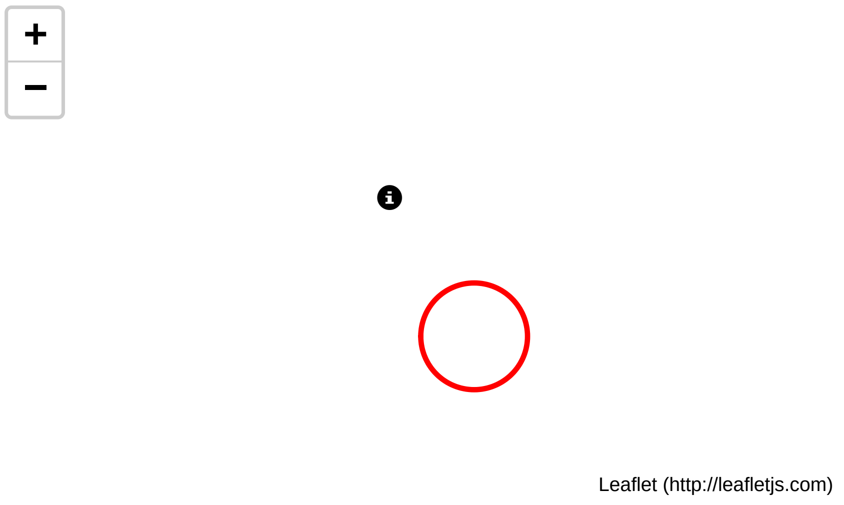
```
markers =
map_hooray.add_child(folium.ClickForMarker(popup="pop_up_name"))
```

Literally add that code to your map and then users can click anywhere to add their own marker.

In [6]:
```
# Set the map up
map_hooray = folium.Map(location=[51.5074, 0.1278],
                        tiles = "Stamen Toner",
                        zoom_start = 9)
# Simple marker
folium.Marker([51.5079, 0.0877],
              popup='London Bridge',
              icon=folium.Icon(color='green')
             ).add_to(map_hooray)

# Circle marker
folium.CircleMarker([51.4183, 0.2206],
                    radius=30,
                    popup='East London',
                    color='red',
                    ).add_to(map_hooray)

# Interactive marker
map_hooray.add_child(folium.ClickForMarker(popup="Dav
e is awesome"))
```

```
map_hooray
```

Out[6]:



Interaction with the map

In [7]:

```
map_hooray = folium.Map(location=[51.5074, 0.1278],
                    zoom_start = 11) # Uses lat then l
on. The bigger the zoom number, the closer in you get

from folium import plugins

# Adds tool to the top right
from folium.plugins import MeasureControl
map_hooray.add_child(MeasureControl())

# Fairly obvious I imagine - works best with transpare
nt backgrounds
from folium.plugins import FloatImage
url = ('https://media.licdn.com/mpr/mpr/shrinknp_100_
100/AAEAAQAAAAAAAAlgAAAAJGE3OTA4YTdlLTkzZjUtNDFjYy1iZ
ThlLWQ5OTNkYzlhNzM4OQ.jpg')
FloatImage(url, bottom=5, left=85).add_to(map_hooray)

map_hooray
```

Out[7]:

float_image
Leaflet (http://leafletjs.com)

Other marker types

I'm skipping over a few markers to move to more interetsing analysis but it's worth knowing that you can also employ. Polygons are markers that let you choose the shape.

folium.RegularPolygonMarker([lat, lon], popup='name', fill_color='color name', number_of_sides= integer, radius=pixels).add_to(map_name)

You can also use Vincent/Vega markers

These are clickable effects. So far, we've just seen text pop-ups. Vincent markers use additional JS to pull in graphical overlays, e.g. click on a pop-up to see the timeline of it's history. You can see an example of them in the Folium documentation, https://folium.readthedocs.io/en/latest/quickstart.html#vincent-vega-markers (https://folium.readthedocs.io/en/latest/quickstart.html#vincent-vega-markers)

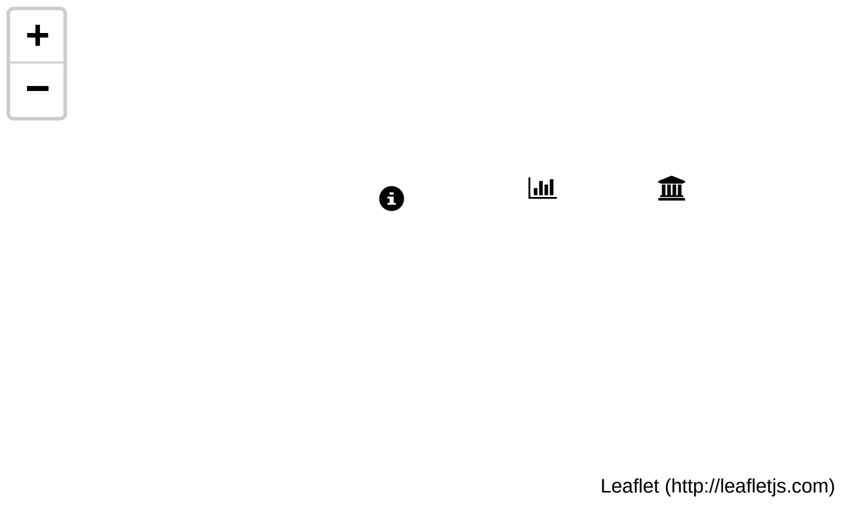
Add icons from fontawesome.io

Reference the "prefix='fa'" to pull icons from fontawesome.io

Run help(folium.Icon) to get the full documentation on what you can do with icons

In [8]:
```
map_hooray = folium.Map(location=[51.5074, 0.1278],
                        tiles = "Stamen Toner",
                        zoom_start = 9)

folium.Marker([51.5079, 0.0877],
              popup='London Bridge',
              icon=folium.Icon(color='green')
             ).add_to(map_hooray)
folium.Marker([51.5183, 0.5206],
              popup='East London',
              icon=folium.Icon(color='red',icon='univ
ersity', prefix='fa')
             ).add_to(map_hooray)

folium.Marker([51.5183, 0.3206]
              popup='East London',
              icon=folium.Icon(color='blue',icon='bar
-chart', prefix='fa')
             ).add_to(map_hooray)
             # icon=folium.Icon(color='red',icon='bicy
cle', prefix='fa')

map_hooray.add_child(folium.ClickForMarker(popup="Dav
e is awesome"))

map_hooray
```

Out[8]:



Heatmaps, boo-ya!

Definitely one of the best functions in Folium. This does not take Dataframes. You'll need to give it a list of lat, lons, i.e. a list of lists. It should be like this. NaNs will also trip it up,

> [[lat, lon],[lat, lon],[lat, lon],[lat, lon],[lat, lon]]

In [9]:

```
from folium import plugins
from folium.plugins import HeatMap


map_hooray = folium.Map(location=[51.5074, 0.1278],
                    zoom_start = 13)

# Ensure you're handing it floats
df_acc['Latitude'] = df_acc['Latitude'].astype(float)
df_acc['Longitude'] = df_acc['Longitude'].astype(floa
t)

# Filter the DF for rows, then columns, then remove Na
Ns
heat_df = df_acc[df_acc['Speed_limit']=='30'] # Reduc
ing data size so it runs faster
heat_df = heat_df[heat_df['Year']=='2007'] # Reducing
 data size so it runs faster
heat_df = heat_df[['Latitude', 'Longitude']]
heat_df = heat_df.dropna(axis=0, subset=['Latitude',
'Longitude'])

# List comprehension to make out list of lists
heat_data = [[row['Latitude'],row['Longitude']] for i
ndex, row in heat_df.iterrows()]

# Plot it on the map
HeatMap(heat_data).add_to(map_hooray)

# Display the map
map_hooray
```

```
Out[9]:
```

Heatmap with time series

This is very similat to Heatmap, just one touch more complicated. It takes a list of list OF LISTS! Yep, another layer deep.

In this example we organise it by month. So we have 12 lists of lists, e.g.

Jen = [[lat,lon],[lat,lon],[lat,lon]] Feb = [[lat,lon],[lat,lon],[lat,lon]] March = [[lat,lon],[lat,lon],[lat,lon]]

list of lists of lists = [Jan, Feb, March] that looks like [[[lat,lon],[lat,lon],[lat,lon]],[[lat,lon], [lat,lon],[lat,lon]],[[lat,lon],[lat,lon],[lat,lon]]]

To understand that better you should use Ctrl+F to spot the double brackets, '[[', and fint eh sub lists.

To make that happen you use a list comprehesntion within a list comprehension. You can see that below where I declare 'heat_data = '

I break this down a little further below the map.

For reasons I don't understand the play, forward, backward buttons are missing their logos but they do work on the bottom left.

In [10]:
```
from folium import plugins

map_hooray = folium.Map(location=[51.5074, 0.1278],
                    zoom_start = 13)

# Ensure you're handing it floats
df_acc['Latitude'] = df_acc['Latitude'].astype(float)
df_acc['Longitude'] = df_acc['Longitude'].astype(floa
t)

# Filter the DF for rows, then columns, then remove Na
Ns
heat_df = df_acc[df_acc['Speed_limit']=='40'] # Reduc
ing data size so it runs faster
heat_df = heat_df[heat_df['Year']=='2007'] # Reducing
 data size so it runs faster
heat_df = heat_df[['Latitude', 'Longitude']]

# Create weight column, using date
heat_df['Weight'] = df_acc['Date'].str[3:5]
heat_df['Weight'] = heat_df['Weight'].astype(float)
heat_df = heat_df.dropna(axis=0, subset=['Latitude',
'Longitude', 'Weight'])

# List comprehension to make out list of lists
heat_data = [[[row['Latitude'],row['Longitude']] for
index, row in heat_df[heat_df['Weight'] == i].iterrow
s()] for i in range(0,13)]

# Plot it on the map
hm = plugins.HeatMapWithTime(heat_data,auto_play=True
,max_opacity=0.8)
hm.add_to(map_hooray)
# Display the map
map_hooray
```

Out[10]:

The lists of lists of lists

a.k.a. list comprehension within a list comprehension)

Here's the line we used.

```
heat_data = [[[row['Latitude'],row['Longitude']] for index, row in
heat_df[heat_df['Weight'] == i].iterrows()] for i in range(0,13)]
```

Rewriting list comprehensions as regular Python can be helpful

```
heat_data1 = []

for i in range(0,13):

    heat_data2 = []

    heat_data1.append(heat_data2)

    for index, row in heat_df[heat_df['Weight'] == i].iter
    rows():

        lt_lon = [row['Latitude'],row['Longitude']]

        heat_data2.append(lat_lon)
```

Plugins

There are too many to demo them all but check out this notebook to see the additional plugins you can use. Likely to be of interest are MarkerCluster and Fullscreen.

http://nbviewer.jupyter.org/github/python-visualization/folium/blob/master/examples/Plugins.ipynb (http://nbviewer.jupyter.org/github/python-visualization/folium/blob/master/examples/Plugins.ipynb)

```
In [11]:
```

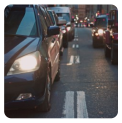
Did you find this Kernel useful?
Show your appreciation with an upvote

6

Data

Data Sources

1.6 million UK tra...
- 35 columns
- 35 columns
- 35 columns
- 29 columns
- accident_coor...
- Areas.shp
- Local_Authorit...



1.6 million UK traffic accidents

Visualise and analyse traffic demographics

Last Updated: a year ago (Version 10)

About this Dataset

Context

The UK government amassed traffic data from 2000 and 2016, recording over 1.6 million accidents in the process and making this one of the most comprehensive traffic data sets out there. It's a huge picture of a country undergoing change.

Note that all the contained accident data comes from police reports, so this data does not include minor incidents.

Content

`ukTrafficAADF.csv` tracks how much traffic there was on all major roads in the given time period (2000 through 2016). AADT, the core statistic included in this file, stands for "Average Annual Daily Flow", and is a measure of how activity a road segment based on how many vehicle trips traverse it. The AADT page on Wikipedia is a good reference on the subject.

Accidents data is split across three CSV files: `accidents_2005_to_2007.csv`, `accidents_2009_to_2011.csv`, and `accidents_2012_to_2014.csv`. These three files together constitute 1.6 million traffic accidents. The total time period is 2005 through 2014, but 2008 is

Run Info

Succeeded	True	Run Time	91.9 secon
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python(Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time   Line #  Log Message
            1  [{
            2    "data": "[NbConvertApp] Converting notebook
               __temp_notebook_source__.ipynb to html\n",
            3    "stream_name": "stderr",
            4    "time": 2.7018236219882965
            5  },{
            6    "data": "[NbConvertApp] Writing 285674 bytes to
               __results__.html\n",
            7    "stream_name": "stderr",
            8    "time": 2.8647348109952873
            9  }{
           10    "data": "[NbConvertApp] Converting notebook
               __temp_notebook_source__.ipynb to notebook\n",
           11    "stream_name": "stderr",
           12    "time": 2.767874495999422
           13  },{
           14    "data": "[NbConvertApp] Executing notebook with
               kernel: python3\n",
           15    "stream_name": "stderr",
           16    "time": 2.7999922669987427
           17  },{
           18    "data": "Fontconfig warning: ignoring C.UTF-8: not
               a valid language tag\n",
           19    "stream_name": "stderr",
           20    "time": 4.071012836007867
           21  },{
           22    "data": "[NbConvertApp] Writing 6810335 bytes to
               __notebook__.ipynb\n",
           23    "stream_name": "stderr",
           24    "time": 20.839875682999264
           25  }{
```

```
  "data": "[NbConvertApp] Converting notebook
__notebook__.ipynb to html\n",
  "stream_name": "stderr",
  "time": 2.7260630179953296
},{
  "data": "[NbConvertApp] Writing 7078018 bytes to
__results__.html\n",
  "stream_name": "stderr",
  "time": 3.0229135019908426
}

Complete. Exited with code 0.
```

Comments (0)

Please sign in to leave a comment.

Our Team Terms Privacy Contact/Support