gmaps Documentation

Release 0.4.0

Pascal Bugnion

January 28, 2017

Contents

CHAPTER 1

Installation

1.1 Dependencies

The current version of *gmaps* is only tested with *IPython 4.2* or later and *ipywidgets 5.1.3* or later. To upgrade to the latest versions, use:

```
$ pip install -U jupyter
```

Make sure that you have enabled widgets extensions to Jupyter:

```
$ jupyter nbextension enable --py --sys-prefix widgetsnbextension
```

1.2 Installing *gmaps*

Install the Python component using:

```
$ pip install gmaps
```

Then tell Jupyter to load the extension with:

```
$ jupyter nbextension enable --py gmaps
```

1.3 Development version

You must have NPM to install the development version. You can install NPM with your package manager.

You must also install `gmaps` in a virtual environment (or, at least, you must be able to run `pip` without root access).

Clone the git repository by running:

```
$ git clone https://github.com/pbugnion/gmaps.git
```

Change to the project's root directory and run:

```
$ pip install -e .
```

This will create a directory called `static/` in the `gmaps/` directory. This directory contains Javascript sources. Every time you change the Javascript sources, you will need to recompile this directory by re-running this command (despite everying being installed in *editable* mode).

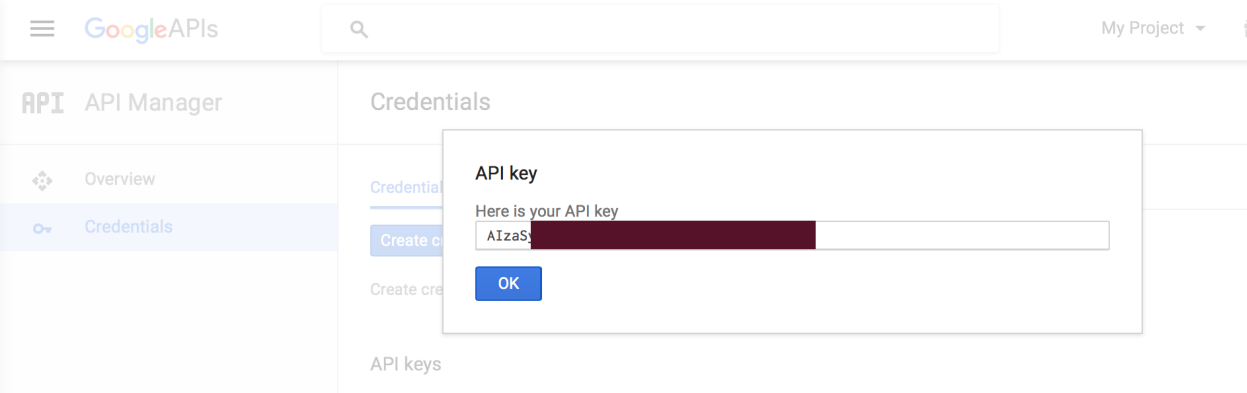
You can then enable the extension in Jupyter:

```
$ jupyter nbextension install --py --symlink --user gmaps
$ jupyter nbextension enable --py --user gmaps
```

CHAPTER 2

Authentication

Most operations on Google Maps require that you tell Google who you are. To authenticate with Google Maps, follow the instructions for creating an API key. You will probably want to create a new project, then click on the *Credentials* section and create a *Browser key*. The API key is a string that starts with the letters `AI`.



You can pass this key to *gmaps* with the `configure` method:

```
gmaps.configure(api_key="AI...")
```

Maps and layers created after the call to `gmaps.configure` will have access to the API key.

You should avoid hard-coding the API key into your Jupyter notebooks. You can use environment variables. Add the following line to your shell start-up file (probably *~/.profile* or *~/.bashrc*):

```
export GOOGLE_API_KEY=AI...
```

Make sure you don't put spaces around the = sign. If you then open a *new* terminal window and type `env` at the command prompt, you should see that your API key. Start a new Jupyter notebook server in a new terminal, and type:

```
import os
import gmaps

gmaps.configure(api_key=os.environ["GOOGLE_API_KEY"])
```

CHAPTER 3

Getting started

gmaps is a plugin for Jupyter for embedding Google Maps in your notebooks. It is designed as a data visualization tool.

To demonstrate *gmaps*, let's plot the earthquake dataset, included in the package:

```
import gmaps
import gmaps.datasets

gmaps.configure(api_key="AI...") # Fill in with your API key

earthquake_df = gmaps.datasets.load_dataset_as_df("earthquakes")
earthquake_df.head()
```

The earthquake data has three columns: a latitude and longitude indicating the earthquake's epicentre and a weight denoting the magnitude of the earthquake at that point. Let's plot the earthquakes on a Google map:

```
locations = earthquake_df[["latitude", "longitude"]]
weights = earthquake_df["magnitude"]
m = gmaps.Map()
m.add_layer(gmaps.heatmap_layer(locations, weights=weights))
m
```

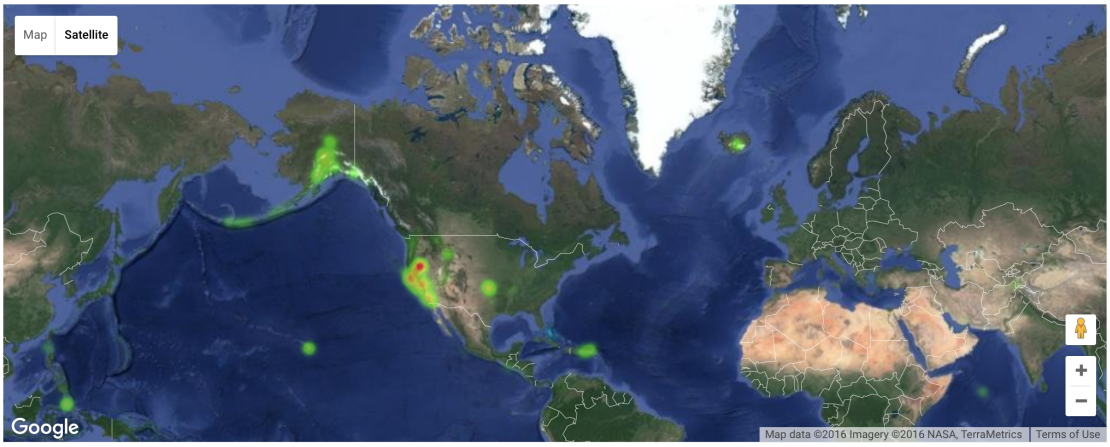
```
In [1]: import gmaps
        import gmaps.datasets
        gmaps.configure(api_key="AIzaSyCTGEf...")
```

```
In [8]: earthquake_df = gmaps.datasets.load_dataset_as_df("earthquakes")
        earthquake_df.head(3)
```

Out[8]:

	latitude	longitude	magnitude
0	65.193300	-149.072500	1.70
1	38.791832	-122.780830	2.10
2	38.818001	-122.792168	0.48

```
In [7]: locations = earthquake_df[["latitude", "longitude"]]
        weights = earthquake_df["magnitude"]
        m = gmaps.Map()
        m.add_layer(gmaps.heatmap_layer(locations, weights=weights))
        m
```



This gives you a fully-fledged Google map. You can zoom in and out, switch to satellite view and even to street view if you really want. The heatmap adjusts as you zoom in and out.

3.1 Basic concepts

gmaps is built around the idea of adding layers to a base map. After you've authenticated with Google maps, you start by creating a base map:

```
import gmaps
gmaps.configure(api_key="AI...")

m = gmaps.Map()
m
```

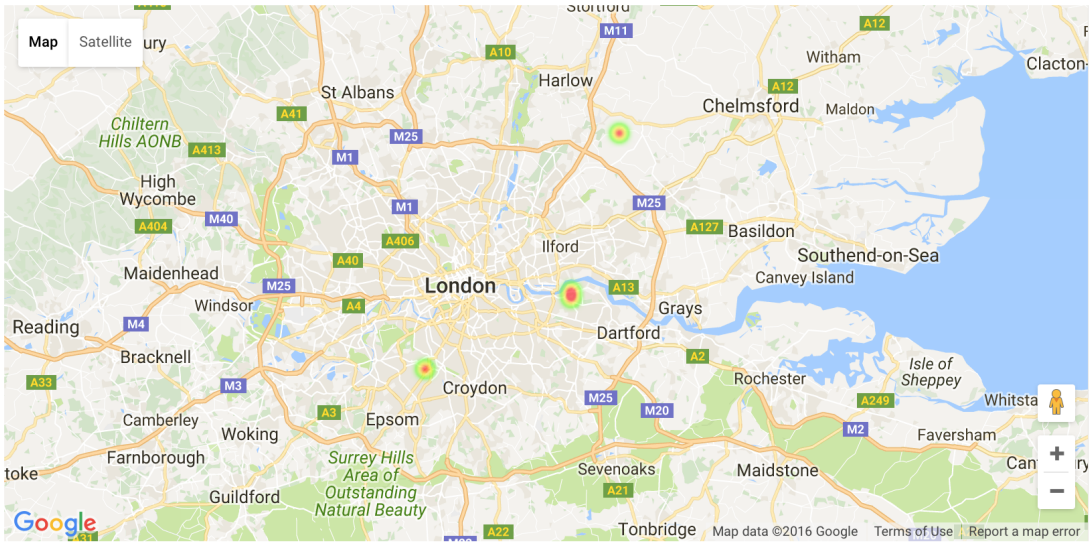
```
In [4]: import gmaps
        import gmaps.datasets
        gmaps.configure("AIzaSyDFgbJoU...")
```

```
In [7]: m = gmaps.Map()
        m
```

You then add layers on top of the base map. For instance, to add a heatmap layer:

```
import gmaps
gmaps.configure(api_key="AI...")

m = gmaps.Map()

# generate some (latitude, longitude) pairs
locations = [(51.5, 0.1), (51.7, 0.2), (51.4, -0.2), (51.49, 0.1)]

heatmap_layer = gmaps.heatmap_layer(locations)
m.add_layer(heatmap_layer)
m
```

```
In [9]: import gmaps
        import gmaps.datasets
        gmaps.configure(api_key="AIzaSyCTGEf...")

        m = gmaps.Map()

        # generate some (latitude, longitude) pairs
        locations = [(51.5, 0.1), (51.7, 0.2), (51.4, -0.2), (51.49, 0.1)]

        heatmap_layer = gmaps.heatmap_layer(locations)
        m.add_layer(heatmap_layer)
        m
```



The *locations* array can either be a list of tuples, as in the example above, a numpy array of shape $N times 2$ or a dataframe with two columns.

Attributes on the base map and the layers can be set through named arguments in the constructor or as instance attributes once the instance is created. These two constructions are thus equivalent:

```
heatmap_layer = gmaps.heatmap_layer(locations)
heatmap_layer.point_radius = 8
```

and:

```
heatmap_layer = gmaps.heatmap_layer(locations, point_radius=8)
```

The former construction is useful for modifying a map once it has been built. Any change in parameters will propagate to maps in which those layers are included.

3.2 Heatmaps

Heatmaps are a good way of getting a sense of the density and clusters of geographical events. They are a powerful tool for making sense of larger datasets. We will use a dataset recording all instances of political violence that occurred in Africa between 1997 and 2015. The dataset comes from the Armed Conflict Location and Event Data Project. This dataset contains about 110,000 rows.:

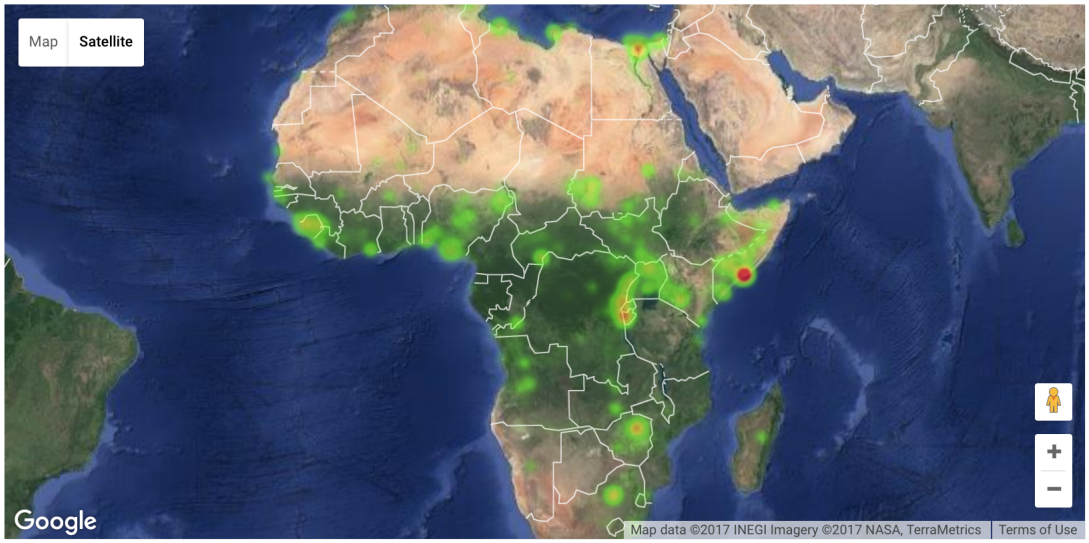
```
import gmaps.datasets

locations = gmaps.datasets.load_dataset_as_df("acled_africa")
```

```
locations.head()
# => dataframe with 'longitude' and 'latitude' columns
```

We already know how to build a heatmap layer:

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...")

locations = gmaps.datasets.load_dataset_as_df("acled_africa")
m = gmaps.Map()
heatmap_layer = gmaps.heatmap_layer(locations)
m.add_layer(heatmap_layer)
m
```

```
In [8]: m = gmaps.Map()

        locations = gmaps.datasets.load_dataset_as_df("acled_africa")
        heatmap_layer = gmaps.heatmap_layer(data)
        m.add_layer(heatmap_layer)
        m
```



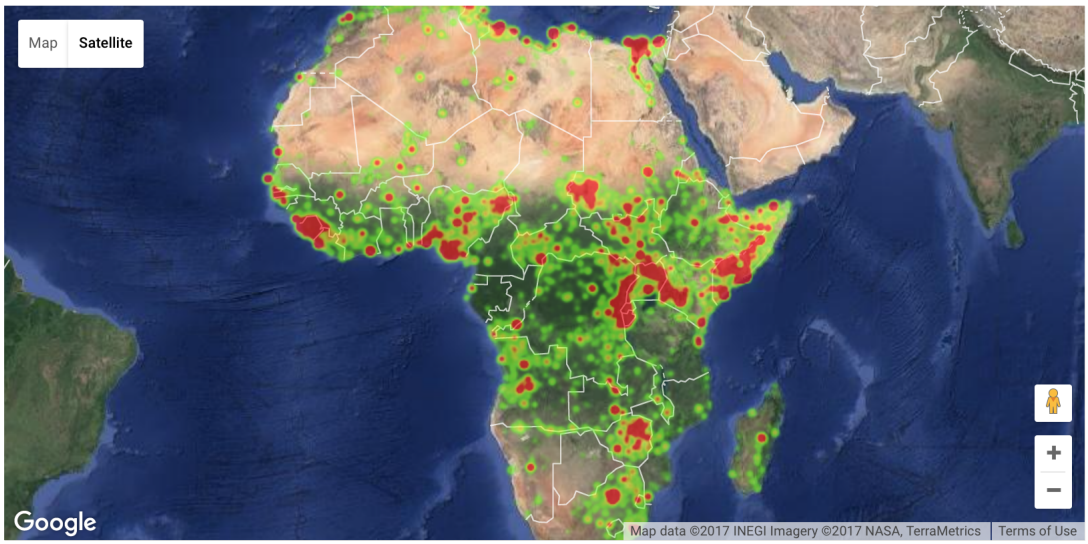
3.2.1 Preventing dissipation on zoom

If you zoom in sufficiently, you will notice that individual points disappear. You can prevent this from happening by controlling the `max_intensity` setting. This caps off the maximum peak intensity. It is useful if your data is strongly peaked. This settings is *None* by default, which implies no capping. Typically, when setting the maximum intensity, you also want to set the `point_radius` setting to a fairly low value. The only good way to find reasonable values for these settings is to tweak them until you have a map that you are happy with.:

```
heatmap_layer.max_intensity = 100
heatmap_layer.point_radius = 5
```

To avoid re-drawing the whole map every time you tweak these settings, you may want to set them in another noteobook cell:

```
In [8]: m = gmaps.Map()

        locations = gmaps.datasets.load_dataset_as_df("acled_africa")
        heatmap_layer = gmaps.heatmap_layer(data)
        m.add_layer(heatmap_layer)
        m
```




```
In [9]: heatmap_layer.max_intensity = 100
        heatmap_layer.point_radius = 5
```

Google maps also exposes a `dissipating` option, which is true by default. If this is true, the radius of influence of each point is tied to the zoom level: as you zoom out, a given point covers more physical kilometres. If you set it to false, the physical radius covered by each point stays fixed. Your points will therefore either be tiny at high zoom levels or large at low zoom levels.

3.2.2 Setting the color gradient and opacity

You can set the color gradient of the map by passing in a list of colors. Google maps will interpolate linearly between those colors. You can represent a color as a string denoting the color (the colors allowed by this):

```
heatmap_layer.gradient = [
    'white',
    'silver',
    'gray'
]
```

If you need more flexibility, you can represent colours as an RGB triple or an RGBA quadruple:

```
heatmap_layer.gradient = [
    (200, 200, 200, 0.6),
    (100, 100, 100, 0.3),
    (50, 50, 50, 0.3)
]
```

```
In [8]: m = gmaps.Map()

        locations = gmaps.datasets.load_dataset_as_df("acled_africa")
        heatmap_layer = gmaps.heatmap_layer(data)
        m.add_layer(heatmap_layer)
        m
```


```
In [11]: heatmap_layer.max_intensity = 100
         heatmap_layer.point_radius = 5
         heatmap_layer.gradient = [
           (200, 200, 200, 0.6),
           (100, 100, 100, 0.3),
           (50, 50, 50, 0.3)
         ]
```

You can also use the `opacity` option to set a single opacity across the entire colour gradient:

```
heatmap_layer.opacity = 0.0 # make the heatmap transparent
```

3.3 Weighted heatmaps

By default, heatmaps assume that every row is of equal importance. You can override this by passing weights through the *weights* keyword argument. The *weights* array is an iterable (e.g. a Python list or a Numpy array) or a single pandas series. Weights must all be positive (this is a limitation in Google maps itself).:

```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AI...")

df = gmaps.datasets.load_dataset_as_df("earthquakes")
# dataframe with columns ('latitude', 'longitude', 'magnitude')

m = gmaps.Map()
heatmap_layer = gmaps.heatmap_layer(
    df[["latitude", "longitude"]], weights=df["magnitude"],
    max_intensity=30, point_radius=3.0
)
m.add_layer(heatmap_layer)
m
```

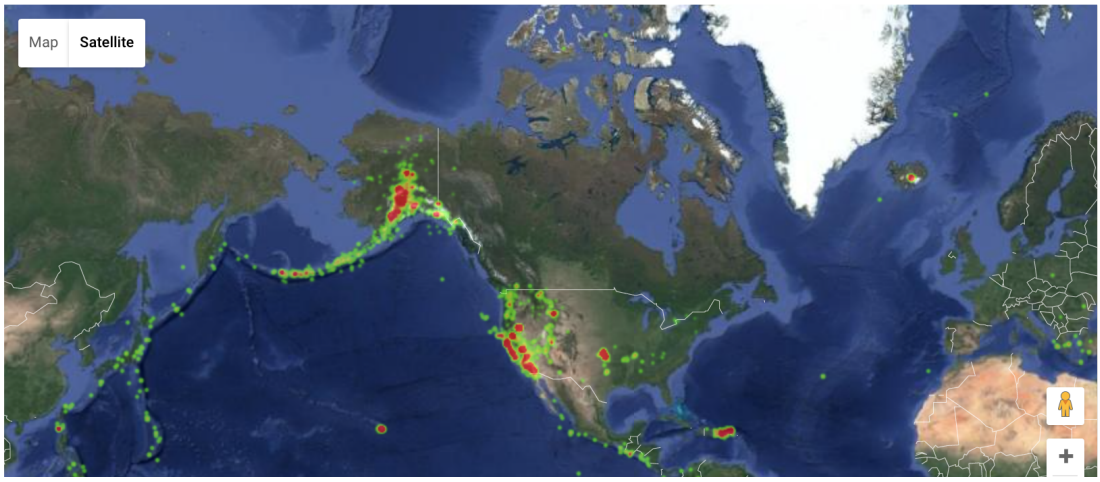
In [ ]:
```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AIza...")
```

In [13]:
```
df = gmaps.datasets.load_dataset_as_df("earthquakes")
df.head()
```

Out[13]:

	latitude	longitude	magnitude
0	65.193300	-149.072500	1.70
1	38.791832	-122.780830	2.10
2	38.818001	-122.792168	0.48
3	33.601667	-116.727667	0.78
4	37.378334	-118.520836	3.64

In [16]:
```
m = gmaps.Map()
heatmap_layer = gmaps.heatmap_layer(
    df[["latitude", "longitude"]], weights=df["magnitude"],
    max_intensity=30, point_radius=3.0
)
m.add_layer(heatmap_layer)
m
```

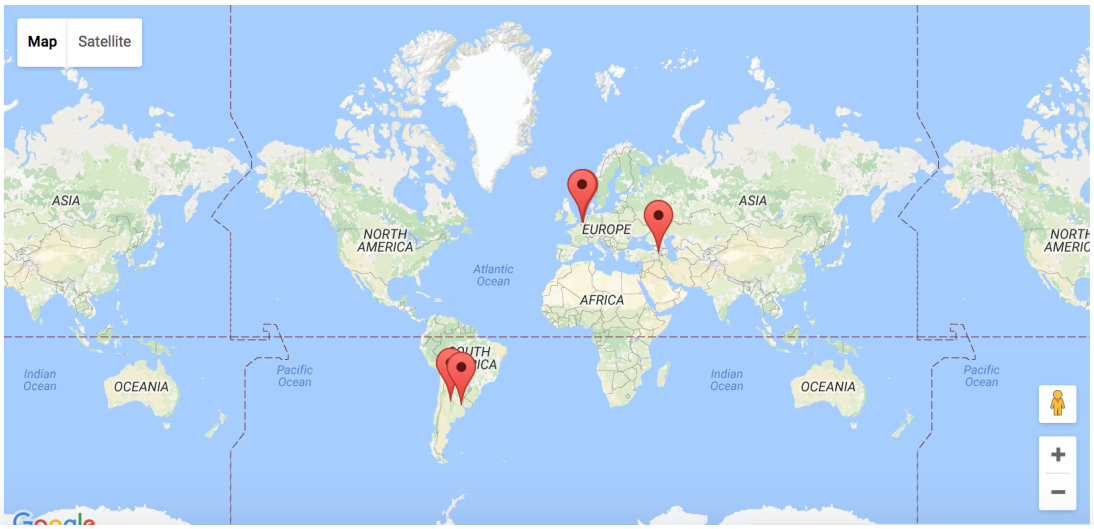


3.4 Markers and symbols

We can add a layer of markers to a Google map. Each marker represents an individual data point:

```
import gmaps
gmaps.configure(api_key="AI...")

marker_locations = [
    (-34.0, -59.166672),
    (-32.23333, -64.433327),
    (40.166672, 44.133331),
    (51.216671, 5.0833302),
    (51.333328, 4.25)
]
markers = gmaps.marker_layer(marker_locations)

m = gmaps.Map()
m.add_layer(markers)
m
```

```
In [8]: marker_locations = [
          (-34.0, -59.166672),
          (-32.23333, -64.433327),
          (40.166672, 44.133331),
          (51.216671, 5.0833302),
          (51.333328, 4.25)
        ]
        markers = gmaps.marker_layer(marker_locations)

        m = gmaps.Map()
        m.add_layer(markers)
        m
```



We can also attach a pop-up box to each marker. Clicking on the marker will bring up the info box. The content of the box can be either plain text or html:

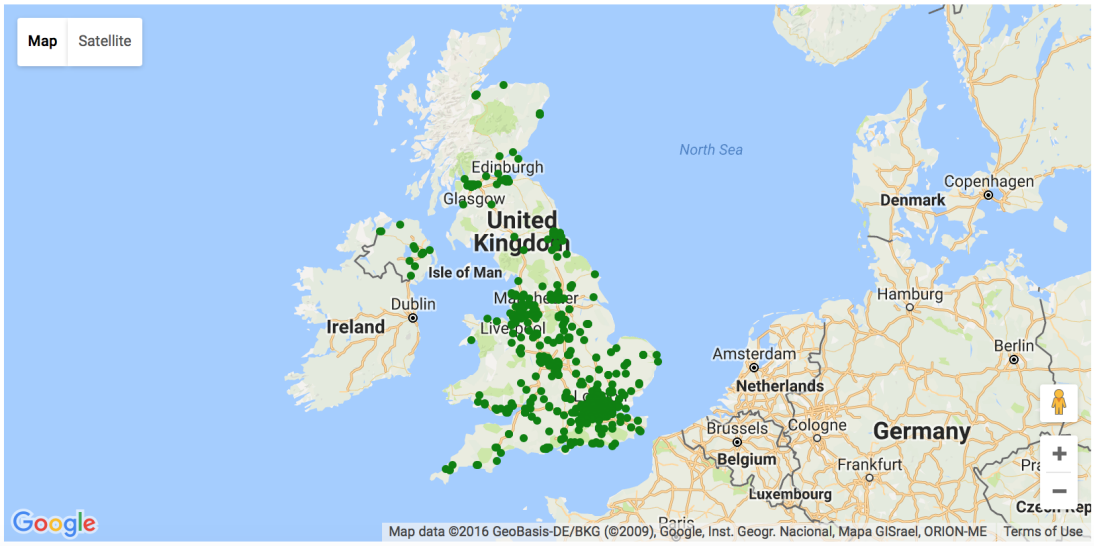
```
import gmaps
gmaps.configure(api_key="AI...")

nuclear_power_plants = [
    {"name": "Atucha", "location": (-34.0, -59.167), "active_reactors": 1},
    {"name": "Embalse", "location": (-32.2333, -64.4333), "active_reactors": 1},
    {"name": "Armenia", "location": (40.167, 44.133), "active_reactors": 1},
    {"name": "Br", "location": (51.217, 5.083), "active_reactors": 1},
    {"name": "Doel", "location": (51.333, 4.25), "active_reactors": 4},
    {"name": "Tihange", "location": (50.517, 5.283), "active_reactors": 3}
]

plant_locations = [plant["location"] for plant in nuclear_power_plants]
info_box_template = """
<dl>
<dt>Name</dt><dd>{name}</dd>
<dt>Number reactors</dt><dd>{active_reactors}</dd>
</dl>
"""
plant_info = [info_box_template.format(**plant) for plant in nuclear_power_plants]

marker_layer = gmaps.marker_layer(plant_locations, info_box_content=plant_info)
m = gmaps.Map()
m.add_layer(marker_layer)
m
```

In [4]:
```
nuclear_power_plants = [
    {"name": "Atucha", "location": (-34.0, -59.167), "active_reactors": 1},
    {"name": "Embalse", "location": (-32.2333, -64.4333), "active_reactors": 1},
    {"name": "Armenia", "location": (40.167, 44.133), "active_reactors": 1},
    {"name": "Br", "location": (51.217, 5.083), "active_reactors": 1},
    {"name": "Doel", "location": (51.333, 4.25), "active_reactors": 4},
    {"name": "Tihange", "location": (50.517, 5.283), "active_reactors": 3}
]

plant_locations = [plant["location"] for plant in nuclear_power_plants]
info_box_template = """
<dl>
<dt>Name</dt><dd>{name}</dd>
<dt>Number reactors</dt><dd>{active_reactors}</dd>
</dl>
"""
plant_info = [info_box_template.format(**plant) for plant in nuclear_power_plants]

marker_layer = gmaps.marker_layer(plant_locations, info_box_content=plant_info)
m = gmaps.Map()
m.add_layer(marker_layer)
m
```

Markers are currently limited to the Google maps style drop icon. If you need to draw more complex shape on maps, use the `symbol_layer` function. Symbols represent each *latitude*, *longitude* pair with a circle whose colour and size you can customize. Let's, for instance, plot the location of every Starbuck's coffee shop in the UK:

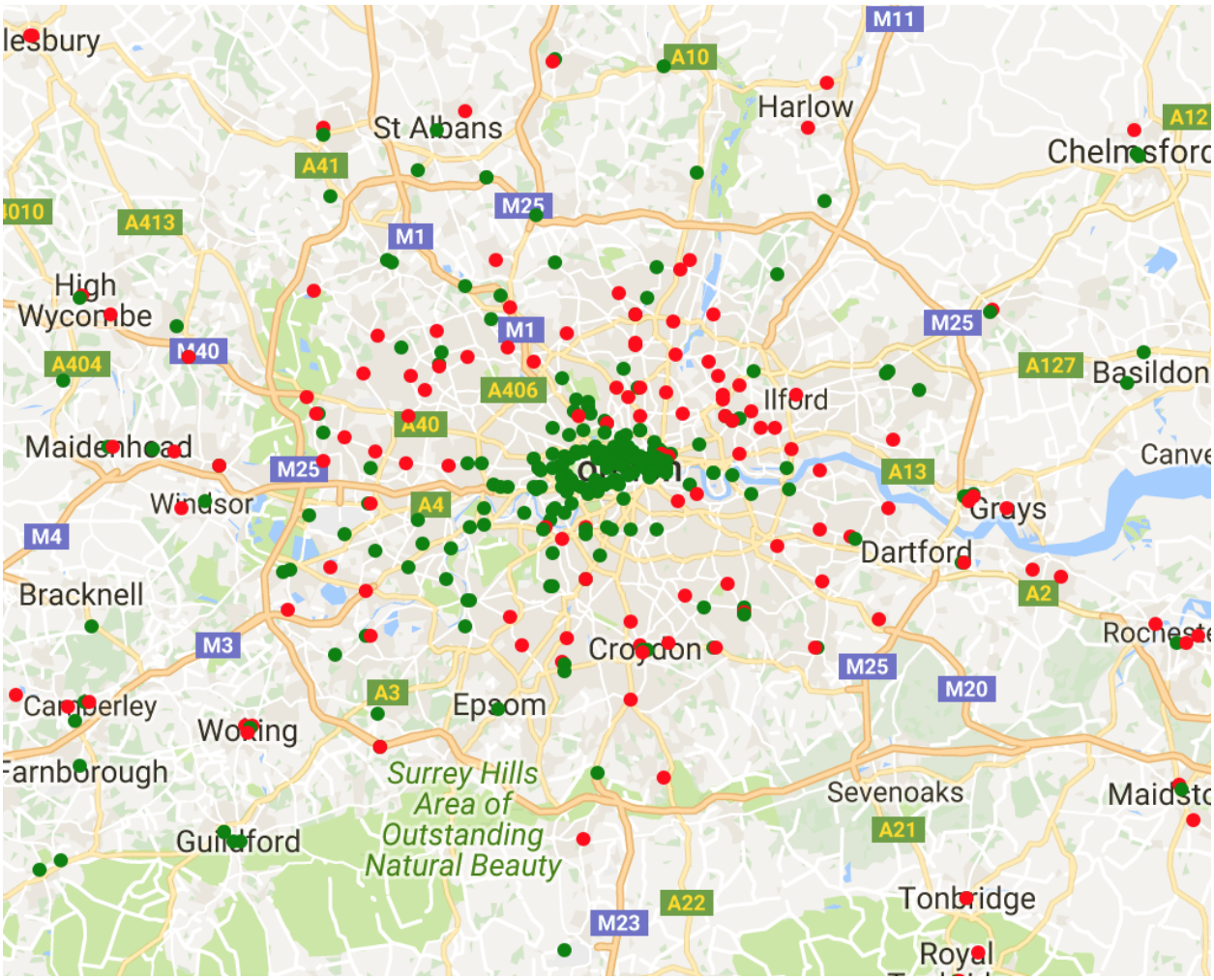
```
import gmaps
import gmaps.datasets

gmaps.configure(api_key="AI...")

starbucks_locations = gmaps.datasets.load_dataset("starbucks_uk")
starbucks_layer = gmaps.symbol_layer(
    starbucks_locations, fill_color="green", stroke_color="green", scale=2
)
m = gmaps.Map()
m.add_layer(starbucks_layer)
m
```

```
In [26]: import gmaps
         import gmaps.datasets

         starbucks_locations = gmaps.datasets.load_dataset("starbucks_uk")
         starbucks_layer = gmaps.symbol_layer(
             starbucks_locations, fill_color="green", stroke_color="green", scale=2
         )
         m = gmaps.Map()
         m.add_layer(starbucks_layer)
         m
```



You can have several layers of markers. For instance, we can compare the locations of Starbucks coffee shops and KFC outlets in the UK by plotting both on the same map:

```
import gmaps
import gmaps.datasets

gmaps.configure(api_key="AI...")

starbucks_locations = gmaps.datasets.load_dataset("starbucks_uk")
kfc_locations = gmaps.datasets.load_dataset("kfc_uk")
starbucks_layer = gmaps.symbol_layer(
    starbucks_locations, fill_color="green", stroke_color="green", scale=2
)
kfc_layer = gmaps.symbol_layer(
    kfc_locations, fill_color="red", stroke_color="red", scale=2
)
m = gmaps.Map()
m.add_layer(starbucks_layer)
m.add_layer(kfc_layer)
m
```



3.4.1 Dataset size limitations

Google maps may become very slow if you try to represent more than a few thousand symbols or markers. If you have a larger dataset, you should either consider subsampling or use heatmaps.

3.5 Directions layer

gmaps supports drawing routes based on the Google maps directions service. At the moment, this only supports driving directions between points denoted by latitude and longitude:

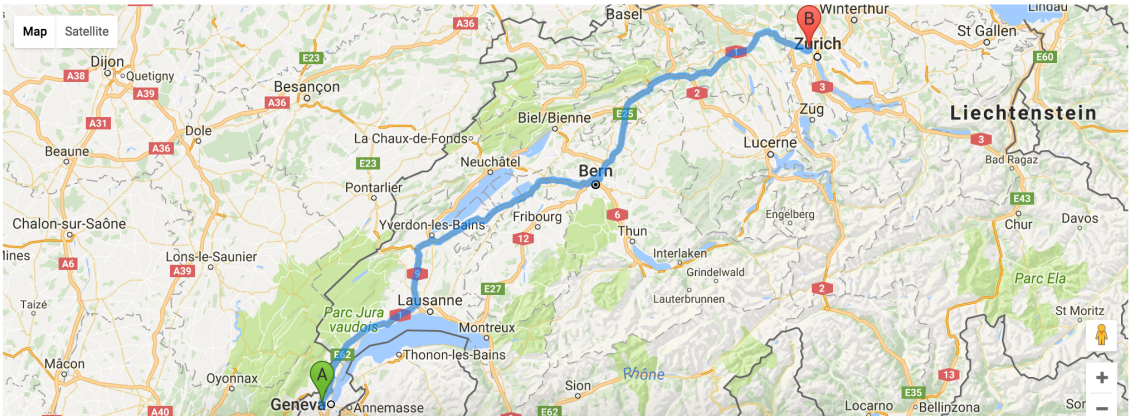
```
import gmaps
import gmaps.datasets
gmaps.configure(api_key="AIza")

# Latitude-longitude pairs
geneva = (46.2, 6.1)
montreux = (46.4, 6.9)
zurich = (47.4, 8.5)

m = gmaps.Map()
```

```
geneva2zurich = gmaps.directions_layer(geneva, zurich)
m.add_layer(geneva2zurich)
m
```

```
In [9]: import gmaps
        import gmaps.datasets
        gmaps.configure(api_key="AIza")
```

```
In [10]: geneva = (46.2, 6.1)
         montreux = (46.4, 6.9)
         zurich = (47.4, 8.5)
```

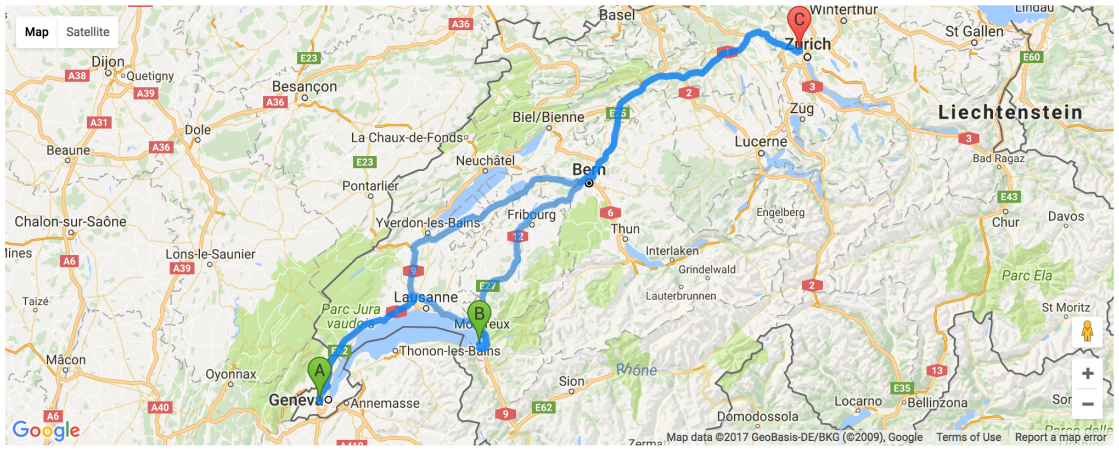
```
In [13]: m = gmaps.Map()
         geneva2zurich_layer = gmaps.directions_layer(geneva, zurich)
         m.add_layer(geneva2zurich_layer)
         m
```



You can also pass waypoints. The can pass up to 23 waypoints (this is a limitation of the Google Maps directions service):

```
m = gmaps.Map()
geneva2zurich_via_montreux =
    gmaps.directions_layer(geneva, zurich, waypoints=[montreux])
m.add_layer(geneva2zurich_via_montreux)
m
```

```
In [12]: m = gmaps.Map()
         geneva2zurich_layer = gmaps.directions_layer(geneva, zurich)
         geneva2zurich_via_montreux_layer = gmaps.directions_layer(geneva, zurich, waypoints=[montreux])
         m.add_layer(geneva2zurich_layer)
         m.add_layer(geneva2zurich_via_montreux_layer)
         m
```



CHAPTER 4

API documentation

4.1 Maps and layers

class gmaps.**Map**(**pargs*, ***kwargs*)

Base map class

Instances of this act as a base map on which you can add additional layers.

Examples

```
>>> m = gmaps.Map()
>>> m.add_layer(gmaps.heatmap_layer(locations))
```

gmaps.**heatmap_layer**(*locations*, *weights=None*, *max_intensity=None*, *dissipating=True*, *point_radius=None*, *opacity=0.6*, *gradient=None*)

Create a heatmap layer.

This returns a `Heatmap` or a `WeightedHeatmap` object that can be added to a `Map` to draw a heatmap. A heatmap shows the density of points in or near a particular area.

To set the parameters, pass them to the constructor or set them on the `Heatmap` object after construction:

```
>>> heatmap_layer = gmaps.heatmap_layer(locations, max_intensity=10)
```

or:

```
>>> heatmap_layer = gmaps.heatmap_layer(locations)
>>> heatmap_layer.max_intensity = 10
```

Examples

```
>>> m = gmaps.Map()
>>> locations = [(46.1, 5.2), (46.2, 5.3), (46.3, 5.4)]
>>> heatmap_layer = gmaps.heatmap_layer(locations)
>>> heatmap_layer.max_intensity = 2
>>> heatmap_layer.point_radius = 3
>>> heatmap_layer.gradient = ['white', 'gray']
>>> m.add_layer(heatmap_layer)
```

Parameters

- **locations** (*iterable of latitude, longitude pairs*) – Iterable of (latitude, longitude) pairs denoting a single point. Latitudes are expressed as a float between -90

(corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and +180 (corresponding to 180 degrees east). This can be passed in as either a list of tuples, a two-dimensional numpy array or a pandas dataframe with two columns, in which case the first one is taken to be the latitude and the second one is taken to be the longitude.

- **weights** (*iterable of floats, optional*) – Iterable of weights of the same length as *locations*. All the weights must be positive.
- **max_intensity** (*float, optional*) – Strictly positive floating point number indicating the numeric value that corresponds to the hottest colour in the heatmap gradient. Any density of points greater than that value will just get mapped to the hottest colour. Setting this value can be useful when your data is sharply peaked. It is also useful if you find that your heatmap disappears as you zoom in.
- **point_radius** (*int, optional*) – Number of pixels for each point passed in the data. This determines the "radius of influence" of each data point.
- **dissipating** (*bool, optional*) – Whether the radius of influence of each point changes as you zoom in or out. If *dissipating* is True, the radius of influence of each point increases as you zoom out and decreases as you zoom in. If False, the radius of influence remains the same. Defaults to True.
- **opacity** (*float, optional*) – The opacity of the heatmap layer. Defaults to 0.6.
- **gradient** (*list of colors, optional*) – The color gradient for the heatmap. This must be specified as a list of colors. Google Maps then interpolates linearly between those colors. Colors can be specified as a simple string, e.g. 'blue', as an RGB tuple, e.g. (100, 0, 0), or as an RGBA tuple, e.g. (100, 0, 0, 0.5).

Returns A *gmaps.Heatmap* or a *gmaps.WeightedHeatmap* widget.

gmaps.**symbol_layer**(*locations*, *hover_text=''*, *fill_color=None*, *fill_opacity=1.0*, *stroke_color=None*, *stroke_opacity=1.0*, *scale=3*, *info_box_content=None*, *display_info_box=None*)

Symbol layer

Add this layer to a `Map` instance to draw symbols on the map. A symbol will be drawn on the map for each point in the `locations` argument.

Examples

```
>>> m = gmaps.Map()
>>> locations = [
        (-34.0, -59.166672),
        (-32.23333, -64.433327),
        (40.166672, 44.133331),
        (51.216671, 5.0833302),
        (51.333328, 4.25)
    ]
>>> symbols = gmaps.symbol_layer(
        locations, fill_color="red", stroke_color="red")
>>> m.add_layer(symbols)
```

You can set a list of information boxes, which will be displayed when the user clicks on a marker.

```
>>> list_of_infoboxes = [
        "Simple string info box",
        "<a href='http://example.com'>HTML content</a>"
    ]
>>> symbol_layer = gmaps.symbol_layer(
            locations, info_box_content=list_of_infoboxes)
```

You can also set text that appears when someone's mouse hovers over a point:

```
>>> names = ["Atucha", "Embalse", "Armenia", "BR", "Doel"]
>>> symbol_layer = gmaps.symbol_layer(locations, hover_text=names)
```

Apart from `locations`, which must be an iterable of (latitude, longitude) pairs, the arguments can be given as either a list of the same length as `locations`, or a single value. If given as a single value, this value will be broadcast to every marker. Thus, these two calls are equivalent:

```
>>> symbols = gmaps.symbol_layer(
        locations, fill_color=["red"]*len(locations))
>>> symbols = gmaps.symbol_layer(
        locations, fill_color="red")
```

The former is useful for passing different colours to different symbols.

```
>>> colors = ["red", "green", "blue", "black", "white"]
>>> symbols = gmaps.symbol_layer(
        locations, fill_color=colors, stroke_color=colors)
```

Parameters

- **locations** (*list of tuples*) – List of (latitude, longitude) pairs denoting a single point. Latitudes are expressed as a float between -90 (corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and +180 (corresponding to 180 degrees east).
- **hover_text** (*string or list of strings, optional*) – Text to be displayed when a user's mouse is hovering over a marker. This can be either a single string, in which case it will be applied to every marker, or a list of strings, in which case it must be of the same length as *locations*. If this is set to an empty string, nothing will appear when the user's mouse hovers over a symbol.
- **fill_color** (*single color or list of colors, optional*) – The fill color of the symbol. This can be specified as a single color, in which case the same color will apply to every symbol, or as a list of colors, in which case it must be the same length as `locations`. Colors can be specified as a simple string, e.g. 'blue', as an RGB tuple, e.g. (100, 0, 0), or as an RGBA tuple, e.g. (100, 0, 0, 0.5).
- **fill_opacity** (*float or list of floats, optional*) – The opacity of the fill color. The opacity should be a float between 0.0 (transparent) and 1.0 (opaque), or a list of floats. 1.0 by default.
- **stroke_color** (*single color or list of colors, optional*) – The stroke color of the symbol. This can be specified as a single color, in which case the same color will apply to every symbol, or as a list of colors, in which case it must be the same length as `locations`. Colors can be specified as a simple string, e.g. 'blue', as an RGB tuple, e.g. (100, 0, 0), or as an RGBA tuple, e.g. (100, 0, 0, 0.5).
- **stroke_opacity** (*float or list of floats, optional*) – The opacity of the stroke color. The opacity should be a float between 0.0 (transparent) and 1.0 (opaque), or a list of floats. 1.0 by default.
- **scale** (*integer or list of integers, optional*) – How large the marker is. This can either be a single integer, in which case the same scale will be applied to every marker, or it must be an iterable of the same length as `locations`. The scale must be greater than 1. This defaults to 3.
- **info_box_content** (*string or list of strings, optional*) – Content to be displayed when user clicks on a marker. This should either be a single string, in which

case the same content will apply to every marker, or a list of strings of the same length of the *locations* list.

- **display_info_box** (*boolean or list of booleans, optional*) – Whether to display an info box when the user clicks on a symbol. This should either be a single boolean value, in which case it will be applied to every symbol, or a list of boolean values of the same length as the *locations* list. The default value is True for any symbols for which *info_box_content* is set, and False otherwise.

gmaps.**marker_layer**(*locations*, *hover_text=''*, *label=''*, *info_box_content=None*, *display_info_box=None*)

Marker layer

Add this layer to a `Map` instance to draw markers corresponding to specific locations on the map. A marker will be drawn on teh map for each point in the `locations` argument.

Examples

```
>>> m = gmaps.Map()
>>> locations = [
        (-34.0, -59.166672),
        (-32.23333, -64.433327),
        (40.166672, 44.133331),
        (51.216671, 5.0833302),
        (51.333328, 4.25)
    ]
>>> symbols = gmaps.marker_layer(locations)
>>> m.add_layer(symbols)
```

Parameters

- **locations** (*list of tuples*) – List of (latitude, longitude) pairs denoting a single point. Latitudes are expressed as a float between -90 (corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and +180 (corresponding to 180 degrees east).
- **hover_text** (*string or list of strings, optional*) – Text to be displayed when a user's mouse is hovering over a marker. This can be either a single string, in which case it will be applied to every marker, or a list of strings, in which case it must be of the same length as *locations*. If this is set to an empty string, nothing will appear when the user's mouse hovers over a marker.
- **label** (*string or list of strings, optional*) – Text to be displayed inside the marker. Google maps only displays the first letter of whatever string is passed to the marker. This can be either a single string, in which case every marker will receive the same label, or a list of strings, in which case it must be of the same length as *locations*.
- **info_box_content** (*string or list of strings, optional*) – Content to be displayed when user clicks on a marker. This should either be a single string, in which case the same content will apply to every marker, or a list of strings of the same length of the *locations* list.
- **display_info_box** (*boolean or list of booleans, optional*) – Whether to display an info box when the user clicks on a marker. This should either be a single boolean value, in which case it will be applied to every marker, or a list of boolean values of the same length as the *locations* list. The default value is True for any markers for which *info_box_content* is set, and False otherwise.

gmaps.**directions_layer**(*start*, *end*, *waypoints=None*)
Create a directions layer.

Add this layer to a Map instance to draw directions on the map. Currently, directions are limited to DRIVING directions.

Examples

```
>>> m = gmaps.Map()
>>> start = (46.2, 6.1)
>>> end = (47.4, 8.5)
>>> directions = gmaps.directions_layer(start, end)
>>> m.add_layer(directions)
>>> m
```

You can also add waypoints on the route:

```
>>> waypoints = [(46.4, 6.9), (46.9, 8.0)]
>>> directions = gmaps.directions_layer(start, end, waypoints=waypoints)
```

Parameters

- **start** (*2-element tuple*) – (Latitude, longitude) pair denoting the start of the journey.
- **end** (*2-element tuple*) – (Latitude, longitude) pair denoting the end of the journey.
- **waypoints** (*List of 2-element tuples, optional*) – Iterable of (latitude, longitude) pair denoting waypoints. Google maps imposes a limitation on the total number of waypoints. This limit is currently 23.

4.2 Utility functions

gmaps.**configure**(*api_key=None*)
Configure access to the GoogleMaps API.

Parameters **api_key** – String denoting the key to use when accessing Google maps, or None to not pass an API key.

gmaps.locations.**locations_to_list**(*locations*)
Convert from a generic iterable of locations to a list of tuples

Layer widgets only accepts lists of tuples, but we want the user to be able to pass in any reasonable iterable. We therefore need to convert the iterable passed in.

4.3 Low level widgets

class gmaps.**Heatmap**(***kwargs*)
Heatmap layer.

Add this to a Map instance to draw a heatmap. A heatmap shows the density of points in or near a particular area.

To set the parameters, pass them to the constructor or set them on the heatmap object after construction:

```
>>> heatmap_layer = gmaps.Heatmap(data=data, max_intensity=10)
```

or:

```
>>> heatmap_layer = gmaps.Heatmap()
>>> heatmap_layer.data = data
>>> heatmap_layer.max_intensity = 10
```

Examples

```
>>> m = gmaps.Map()
>>> data = [(46.1, 5.2), (46.2, 5.3), (46.3, 5.4)]
>>> heatmap_layer = gmaps.Heatmap(data=data)
>>> heatmap_layer.max_intensity = 2
>>> heatmap_layer.point_radius = 3
>>> heatmap_layer.gradient = ['white', 'gray']
>>> m.add_layer(heatmap_layer)
```

Parameters

- **data** (*list of tuples*) – List of (latitude, longitude) pairs denoting a single point. Latitudes are expressed as a float between -90 (corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and 180 (corresponding to 180 degrees east).
- **max_intensity** (*float, optional*) – Strictly positive floating point number indicating the numeric value that corresponds to the hottest colour in the heatmap gradient. Any density of points greater than that value will just get mapped to the hottest colour. Setting this value can be useful when your data is sharply peaked. It is also useful if you find that your heatmap disappears as you zoom in.
- **point_radius** (*int, optional*) – Number of pixels for each point passed in the data. This determines the "radius of influence" of each data point.
- **dissipating** (*bool, optional*) – Whether the radius of influence of each point changes as you zoom in or out. If *dissipating* is True, the radius of influence of each point increases as you zoom out and decreases as you zoom in. If False, the radius of influence remains the same. Defaults to True.
- **opacity** (*float, optional*) – The opacity of the heatmap layer. Defaults to 0.6.
- **gradient** (*list of colors, optional*) – The color gradient for the heatmap. This must be specified as a list of colors. Google Maps then interpolates linearly between those colors. Colors can be specified as a simple string, e.g. 'blue', as an RGB tuple, e.g. (100, 0, 0), or as an RGBA tuple, e.g. (100, 0, 0, 0.5).

class gmaps.**WeightedHeatmap**(***kwargs*)

Heatmap with weighted points.

Add this layer to a Map instance to draw a heatmap. Unlike the plain Heatmap layer, which assumes that all points should have equal weight, this layer lets you specifiy different weights for points.

Examples

```
>>> m = gmaps.Map()
# triples representing `latitude, longitude, weight`:
>>> data = [(46.1, 5.2, 0.5), (46.2, 5.3, 0.2), (46.3, 5.4, 0.8)]
>>> heatmap_layer = gmaps.Heatmap(data=data)
>>> heatmap_layer.max_intensity = 2
>>> m.add_layer(heatmap_layer)
```

Parameters

- **data** (*list of tuples*) – List of (latitude, longitude, weight) triples for a single point. Latitudes are expressed as a float between -90 (corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and +180 (corresponding to 180 degrees east). Weights must be non-negative.
- **max_intensity** (*float, optional*) – Strictly positive floating point number indicating the numeric value that corresponds to the hottest colour in the heatmap gradient. Any density of points greater than that value will just get mapped to the hottest colour. Setting this value can be useful when your data is sharply peaked. It is also useful if you find that your heatmap disappears as you zoom in.
- **point_radius** (*int, optional*) – Number of pixels for each point passed in the data. This determines the "radius of influence" of each data point.
- **dissipating** (*bool, optional*) – Whether the radius of influence of each point changes as you zoom in or out. If *dissipating* is True, the radius of influence of each point increases as you zoom out and decreases as you zoom in. If False, the radius of influence remains the same. Defaults to True.
- **opacity** (*float, optional*) – The opacity of the heatmap layer. Defaults to 0.6.
- **gradient** (*list of colors, optional*) – The color gradient for the heatmap. This must be specified as a list of colors. Google Maps then interpolates linearly between those colors. Colors can be specified as a simple string, e.g. 'blue', as an RGB tuple, e.g. (100, 0, 0), or as an RGBA tuple, e.g. (100, 0, 0, 0.5).

class gmaps.**Symbol**(***kwargs*)
Class representing a single symbol.

Symbols are like markers, but the point is represented by an SVG symbol, rather than the default inverted droplet. Symbols should be added to the map via the 'Markers' widget.

class gmaps.**Marker**(***kwargs*)
Class representing a marker.

Markers should be added to the map via the 'Markers' widget.

class gmaps.**Markers**(***kwargs*)
A collection of markers or symbols.

class gmaps.**Directions**(***kwargs*)
Directions layer.

Add this to a `Map` instance to draw directions.

The directions are requested with the `DRIVING` option.

Data is a list of latitude, longitude tuples. The first point of the list is passed as the origin of the itinerary and the last point is passed as the destination of the itinerary. Other points are passed in order as a list of waypoints.

Examples

```
>>> m = gmaps.Map()
>>> data = [(48.85341, 2.3488), (50.85045, 4.34878), (52.37403, 4.88969)]
>>> directions_layer = gmaps.Directions(data=data)
>>> m.add_layer(directions_layer)
```

There is a limitation in the number of waypoints allowed by Google (currently 23). If it fails to return directions, a DirectionsServiceException is raised.

```
>>> directions_layer = gmaps.Directions(data=data*10)
Traceback (most recent call last):
    ...
DirectionsServiceException: No directions returned: MAX WAYPOINTS EXCEEDED
```

Parameters **data** (*list of tuples of length >= 2*) – List of (latitude, longitude) pairs denoting a single point. The first pair denotes the starting point and the last pair denote the end of the route. Latitudes are expressed as a float between -90 (corresponding to 90 degrees south) and +90 (corresponding to 90 degrees north). Longitudes are expressed as a float between -180 (corresponding to 180 degrees west) and 180 (corresponding to 180 degrees east).

4.4 Datasets

gmaps.datasets.**list_datasets**()
List of datasets available

gmaps.datasets.**load_dataset**(*dataset_name*)
Fetch a dataset, returning an array of tuples.

gmaps.datasets.**dataset_metadata**(*dataset_name*)
Information about the dataset

This returns a dictionary containing a 'description', a list of the dataset headers and optionally information about the dataset source.

Examples

```
>>> dataset_metadata("earthquakes")
{'description': 'Taxi pickup location data in San Francisco',
 'headers': ['latitude', 'longitude']}
```

gmaps.datasets.**load_dataset_as_df**(*dataset_name*)
Fetch a dataset, returning a pandas dataframe.

4.5 Traitlets

class gmaps.geotraitlets.**ColorAlpha**(*default_value=traitlets.Undefined*, *allow_none=False*, ***metadata*)
Trait representing a color that can be passed to Google maps.

This is either a string like 'blue' or '#aabbcc' or an RGB tuple like (100, 0, 250) or an RGBA tuple like (100, 0, 250, 0.5).

validate(*obj*, *value*)
Verifies that 'value' is a string or tuple and converts it to a value like 'rgb(x,y,z)'

class gmaps.geotraitlets.**ColorString**(*default_value=traitlets.Undefined*, *allow_none=False*, *read_only=None*, *help=None*, ***kwargs*)
A string holding a color recognized by Google Maps.

Apparently Google Maps accepts 'all CSS3 colors, including RGBA, [...] except for extended named colors and HSL(A) values'.

Using *this <https://www.w3.org/TR/css3-color/#html4>* page for reference.

default_value = traitlets.Undefined

class gmaps.geotraitlets.**Latitude**(*default_value=traitlets.Undefined*, *allow_none=False*, ***kwargs*)

Float representing a latitude

Latitude values must be between -90 and 90.

default_value = traitlets.Undefined

class gmaps.geotraitlets.**Longitude**(*default_value=traitlets.Undefined*, *allow_none=False*, ***kwargs*)

Float representing a longitude

Longitude values must be between -180 and 180.

default_value = traitlets.Undefined

class gmaps.geotraitlets.**Point**(*default_value*)

Tuple representing a (latitude, longitude) pair.

CHAPTER 5

Contributing to jupyter-gmaps

Contributions are welcome! If you're not sure how to get started, or want to run some ideas past the committers, open an issue through the *Github issue tracker <https://github.com/pbugnion/gmaps/issues>*.

5.1 How to release jupyter-gmaps

This is a set of instructions for releasing to Pypi.

- Append the suffix `rc1` to the version in `_version.py`
- Upload the pre-release to Pypi with `python setup.py sdist upload`. Unfortunately, Pypi does not recognize this as a pre-release, and therefore gives it more precendence than the previous, stable release. To correct this, go to the gmaps page on Pypi, then go to the *releases* tab and manually hide that release and un-hide the previous one.
- Verify that you can install the new version and that it works correctly with `pip install gmaps==<new version>` and `jupyter nbextension enable --py --sys-prefix gmaps`. It's best to verify the installation on a clean virtual machine (rather than just in a new environment) since installation is more complex than for pure Python packages.
- If the manual installation tests failed, fix the issue and repeat the previous steps with `rc2` etc. If installing worked, proceed to the next steps.
- Write the changelog for the new version and commit the changes.
- Bump the version number in `_version.py` to a stable version (e.g. 0.3.6).
- Bump the version number in `docs/source/conf.py` for both the `version` and the `release` variables.
- Bump the version number in `js/package.json`.
- Run `python setup.py sdist bdist upload` to upload the artefact to pypi.
- Verify that the new version is available by running `pip install gmaps` in a new virtual environment.
- Commit the changes in version.
- Tag the new version with an annotated tag, e.g. `git tag -a v0.3.6`. Include a `v` in front of the version number. Copy the release notes as the tag annotation.
- Push the new commits and the tag with `git push origin master --tags`
- Change the version number in `_version.py` back to a `dev` version. It's better to bump just the patch release, even if you think the next release may be a minor release.
- Commit the change with a message like *Back to dev*.

- Push the change to master.

CHAPTER 6

Release notes

6.1 Version 0.4.0

- **Add factory functions to make creating layers easier. Instead of creating widgets directly, the widgets are instantiated thro**
  - passing arbitrary iterables to the factory function (issue #66)
  - passing more complex sets of options (issue #65)
- The directions interface is now a first class layer (issue #64)
- A regression whereby the API documentation wasn't building on readthedocs is now fixed (PR #105).

6.2 Version 0.3.6

- Adds info boxes to the marker and symbol layers (PR #98).

6.3 Version 0.3.5

- Bugfix in deprecated heatmap method (PR #89).

6.4 Version 0.3.4

- Add marker and symbol layer (PR #78)
- Fix bug involving incorrect latitude bound calculation.

6.5 Version 0.3.3

- Improve automatic bounds calculations for heatmaps (PR #84)

6.6 Version 0.3.2

- Allow setting heatmap options (issues #74)
- Basic unit tests for traitlets, mixins and datasets
- Continuous integration with Travis CI.

6.7 Version 0.3.1

Fix release to allow injecting Google maps API keys. Google maps now mandates API keys, so this release provides a way to pass in a key (issue #61).

This release also includes a fix for having multiple layers on the same map.

6.8 Version 0.3.0

Complete re-write of gmaps to work with IPython 4.2 and ipywidgets 5.x. This release is at feature parity with the previous release, but the interface differs:

- Maps are now built up from a base to which we add layers.
- Heatmaps and weighted heatmaps are now layers that can be added to the base map.
- Add the acled_africa dataset to demonstrate heatmaps with a substantial amount of data.
- Now fits into the Jupyter installation convention for widget extensions.
- Add sphinx documentation
- Remove example notebooks (these may be added back in a later release)

6.9 Version 0.2.2

- Remove dependency on Numpy
- Fix broken datasets example (issue #52)

6.10 Version 0.2.1

test release – no changes.

6.11 Version 0.2

- IPython 4.0 compatibility
- Python 3 compatibility
- Drop IPython 2.x compatibility

6.12 Version 0.1.6

Fixed typo in setup script.

6.13 Version 0.1.5

Weighted heatmaps and datasets

- Added possibility of including weights in heatmap data.
- Added a datasets module to allow new users to play around with data without having to find their own dataset.

6.14 Version 0.1.4

Another bugfix release.

- Fixed a bug that arose when using heatmap with default values of some of the parameters.

6.15 Version 0.1.3

Bugfix release.

- Fixed a bug that arose when using the heatmap with IPython2.3 in the previous release. The bug was caused by the slightly different traitlets API between the two IPython versions.

6.16 Version 0.1.2

Minor heatmap improvements.

- Exposed the 'maxIntensity' and 'radius' options for heatmaps.

6.17 Version 0.1.1

Bugfix release.

- Ensures the notebook extensions are actually included in the source distribution.

6.18 Version 0.1

Initial release.

- Allows plotting heatmaps from a list / array of pairs of longitude, latitude floats on top of a Google Map.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

g

C

D

G

H

L

M

P

S

V

W