Let’s make a map! Using Geopandas, Pandas and Matplotlib to make a Choropleth map

Benjamin Cooley

Jun 25, 2018 · 6 min read

So you want to make a map using Python. Let’s get started!

If you’ve started doing some data visualisation with Matplotlib and Pandas, but are looking for the next simple step to getting started with geographical data, I got you. I’ve been there. In fact, I spent

hours trawling through online tutorials looking for the easiest package to get started with making maps (specifically choropleths). And while there are lots of options to choose from, I eventually landed on Geopandas as the lowest barrier of entry.

Geopandas is great, cause it's just like Pandas (but using geodata from things like shape files). Geopandas dataframes are a lot like Pandas dataframes, so the two usually play nicely. Best of all, Geopandas allows you to create quick, standalone choropleth maps without many other dependencies (and not too many lines of code!).

Here's a preview of what we'll be making.

It's London! Made with Python.

As an aside, there are lots and lots of great ways to make maps out there (notably, Datawrapper just added a GeoJson wrapper to load

your own custom maps). There's no one size fits all. However, most of these services come with some kind of restriction (like not being able to download a file as svg. Also, making maps in Python give you a couple unique benefits:

- **Reproducibility** — always a strong selling point with Python, but especially for making super quick charts. This tutorial will streamline the process of creating a map as much as possible (using global variables, cleaning, etc) so that next time you want to make a map, you just need to change the csv file (assuming it's the same geographic location).
- **Maximum control** — customise, download in whatever format you want, you make the call. Even though it can take some fiddling with the code, Matplotlib is massively powerful.
- **Lots and lots of maps** — if you need to visualise the same map with lots of variables (small multiple maps?) or maps showing change over time, wrapping this code in a for loop is a smart way to go (which I'll cover in my next post). Chart builders with a GUI interface are great, but usually not so good at automating tasks. Python is Very Good.
- **No design skills needed** — well, almost no design skills. An eye for good data design is helpful. But no Adobe Illustrator or Photoshop skills needed.

Ok, let's do this. Here's what you'll need. I use a Jupyter Notebook to house all the code (which I highly recommend so you can preview rendering), but you do you.

- Pandas
- Geopandas
- Matplotlib

That's it!

Getting the data

Let's get some data into our Notebook. As I'm based in London currently, I'll be making a map of London by local ward (borough level). The London Datastore does a great job making lots of data public and accessible, and I found this page with a bunch of shape files with different levels of detail. Nice!

Click → download → Save as → Move to local directory of notebook. Nailed it.

But the shapefile is only one layer of data. This will help to draw the map, but if we want to bind data to it we will need another dataset as well. Back to London Datastore: let's download the London borough profiles dataset as a csv (which is already pre-cleaned and tidy). This csv file has lots of columns that we can use as variables to visualise.

Now that both datasets are ready to go, I'm back in my Jupyter Notebook. Time to load in the .shp and .csv file.

```
# set the filepath and load in a shapefile

fp = "datasets/geo-data/gis-boundaries-
```

```
london/ESRI/London_Borough_Excluding_MHW.shp"

map_df = gpd.read_file(fp)

# check data type so we can see that this is not a
normal dataframe, but a GEOdataframe

map_df.head()
```

Now let's preview what our map looks like with no data in it.

```
map_df.plot()
```

Cool, it's London!

And then let's load in a csv file of data to join with the geodataframe.

```
df = pd.read_csv("datasets/london-borough-
profile.csv", header=0)

df.head()
```

Cleaning and joining dataframes

Great. So now we have two dataframes ready to go. Let's get a slice of the data that we are going to use.

```
df = df[['borough','Happiness_score_2011-
14_(out_of_10)', 'Anxiety_score_2011-
14_(out_of_10)',
'Population_density_(per_hectare)_2017',
'Mortality_rate_from_causes_considered_preventable_
2012/14']]
```

Those are really terrible column names. Let's rename them to something simpler.

```
data_for_map = df.rename(index=str, columns=
{"Happiness_score_2011-14_(out_of_10)":
"happiness",

"Anxiety_score_2011-14_(out_of_10)": "anxiety",

"Population_density_(per_hectare)_2017":
"pop_density_per_hectare",

"Mortality_rate_from_causes_considered_preventable_
2012/14": 'mortality'})

# check dat dataframe
```

```
data_for_map.head()
```


Much better. Now we need to merge our geodata with our cleaned London dataset. We'll do that using pd.join().

```
# join the geodataframe with the cleaned up csv
dataframe

merged =
map_df.set_index('NAME').join(data_for_map.set_inde
x('borough'))

merged.head()
```

Map time!

Let's start mapping. First we need to do some prep work for Matplotlib. We'll start by setting a variable to map, setting the range and creating the figure for the map to be drawn in.

```
# set a variable that will call whatever column we
want to visualise on the map
```

```
variable = ‘pop_density_per_hectare’

# set the range for the choropleth

vmin, vmax = 120, 220

# create figure and axes for Matplotlib

fig, ax = plt.subplots(1, figsize=(10, 6))
```

The stage has been set. Map time.

```
# create map

merged.plot(column=variable, cmap=’Blues’,
linewidth=0.8, ax=ax, edgecolor=’0.8')
```

And there it is! Not perfect. A little bit warped, and there’s a weird axis around the whole thing that doesn’t really mean anything. But

we have a choropleth. Now let's do some prettifying to get it looking fresh.

Customising the map

First off, that axis needs to go.

```
# remove the axis

ax.axis('off')
```

Then let's add a title to our map, and some text noting the source. Maps are usually nice to look at, but if you don't provide context then it doesn't mean much.

```
# add a title
```

```
ax.set_title('Preventable death rate in London',
fontdict={'fontsize': '25', 'fontweight' : '3'})

# create an annotation for the data source

ax.annotate('Source: London Datastore, 2014',xy=
(0.1, .08),  xycoords='figure fraction',
horizontalalignment='left',
verticalalignment='top', fontsize=12,
color='#555555')
```

Beautiful! Still one thing missing though. We should probably add a legend that shows the range of value for the user. This will help it to not look so squished as well.

```
# Create colorbar as a legend

sm = plt.cm.ScalarMappable(cmap='Blues',
```

```
norm=plt.Normalize(vmin=vmin, vmax=vmax))

# empty array for the data range

sm._A = []

# add the colorbar to the figure

cbar = fig.colorbar(sm)
```

Map. Made.

Last thing: we need to save the map so we can post a tweet with that sweet #dataviz hashtag.

Matplotlib gives you lots of freedom in how you save figures. The code below will save the figure as a png, but if you want to fiddle about some more with it in Illustrator you can also save as svg. If you save as png, make sure to use a dpi of 200 or above. Otherwise

the map and text will look all blurry. Nobody wants that.

```
fig.savefig("map_export.png", dpi=300)
```

And now, we have a publish-ready map waiting in our working directory! Amazing.

So that's it. You're all setup with Geopandas (at least for choropleths). Next I'll look at how to make multiple maps using Geopandas and turn it into a cool gif map. Stay tuned!

You can view and download my Jupyter Notebook for this tutorial here on Github.

Data Science | Data Visualization | Python | Maps | Data Analysis