Synoptic Meteorology I: Assignment #6

Due: 30 October 2018

Learning Objectives: to learn how Python tools can be used to create a map, and how to overlay data and/or geographic information on such a map.

Meteorological data such as surface or upper-air isobaric observations are often viewed spatially. Although there is not yet a single map module around which the Python community has coalesced, the UK Met Office has developed a module called cartopy that allows for a wide range of maps to be created to aid in data visualization. In this assignment, we will cover the basics of how to create a map using cartopy and how to use Python to overlay fixed data on that map. To begin, follow the instructions given in Assignment #5 to create a new script in Spyder.

As with all Python modules, we need to begin our script with the necessary import statements. We have two cartopy functions to import:

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

The first, `crs`, allows us to establish the map projection. The second, `feature`, allows us to add selected geographic features to the map established through a call to the `crs` function.

Since cartopy builds on the matplotlib `pyplot` plotting routines, we must also import that module:

```
import matplotlib.pyplot as plt
```

We begin by setting up our figure in a similar fashion to in Assignment #5:

```
fig = plt.subplots(figsize=(16,9))
```

Note that we only initialize `fig` here rather than both `fig` and `ax`. This is because we initialize `ax` separately to be able to use it to define the map projection:

```
ax = plt.axes(projection=ccrs.PlateCarree())
```

Here, we have initialized a map using a Plate Carree map projection, an equirectangular projection that maps lines of latitude and longitude to straight vertical and horizontal lines, respectively, with equal spacing. It does not maintain areas, nor is it a conformal projection, but it is useful for basic visualizations. It is also one of the easiest in Cartopy to work with, motivating its use here.

By default, the resulting map covers a large range of latitudes and longitudes. We can narrow our focus by specifying the `set_extent` attribute to the axes:

```
ax.set_extent([-100, -80, 35, 50])
```

This sets the map's extent to 80-100°W, 35-50°N, or approximately the Midwest United States.

Now that we have established a blank map, we want to add some of the basic information that we expect a map to convey – namely, geographic references. Cartopy provides a built-in interface to the Natural Earth database that provides one way by which such data can be added to the map. For

example, the following four commands define state/province borders, locations that are on land, and locations that are on inland lakes and major ocean bodies, respectively:

```
states = cfeature.NaturalEarthFeature(category='cultural',
          name='admin_1_states_provinces_lines', scale='10m',
          facecolor='none', edgecolor='grey', linewidth=1)
land = cfeature.NaturalEarthFeature(category='physical',
          name='land', scale='10m', edgecolor='grey',
          facecolor=cfeature.COLORS['land'])
lakes = cfeature.NaturalEarthFeature(category='physical',
          name='lakes', scale='10m', edgecolor='black',
          facecolor=cfeature.COLORS['water'])
oceans = cfeature.NaturalEarthFeature(category='physical',
          name='ocean,' scale='50m', edgecolor='black',
          facecolor=cfeature.COLORS['water'])
```

The `feature` function's `NaturalEarthFeature` attribute is invoked to define the variables that contain the desired geographic information. There are two types of categories, cultural for all human-defined features and physical for natural features. The `name` keyword refers to the name given to a dataset within the Natural Earth database; these are obtained by browsing the Natural Earth website (https://www.naturalearthdata.com/downloads/). The `scale` keyword refers to the detail provided by the data; 10m is the highest-resolution available and represents a 1:10,000,000 map scale. The `edgecolor` and `facecolor` keywords define the color given to boundary edges and to boundary interiors, respectively. These can be a matplotlib defined color or a cartopy feature defined color, and examples of each are given. Finally, the `linewidth` keyword defines the line width for the chosen field.

Once we have defined these features, we need to add them to the axes:

```
ax.add_feature(land)
ax.add_feature(lakes)
ax.add_feature(states)
ax.add_feature(oceans)
```

However, these features do not include national or coastline boundaries. We have to add these with separate arguments:

```
ax.add_feature(cfeature.BORDERS, edgecolor='grey')
          ax.coastlines(resolution='10m')
```

Note that there is an intrinsic cartopy feature called `BORDERS` to provide national boundaries, so we do not have to reference more Natural Earth data for this task.

Enter the above code into your script and run it. It may take a few minutes to download the needed geographic data the first time you do so. Once done, you should get a nice map of the Midwestern United States and adjacent portions of southern Canada.

Now that we have a nice-looking map, we can add some fixed data to it. For instance, let's say that we have a few Python lists containing IDs, latitudes, and longitudes of Wisconsin's NWS offices:

```
locations = ['MKX', 'GRB', 'ARX']
  lats = [42.97, 44.50, 43.82]
 lons = [-88.55, -88.11, -91.19]
```

Let's say we want to add markers at each office's location and a text label beneath the marker with each office's ID. We can do this using Python's `for` looping structure and a couple of matplotlib plotting routines:

```
for i in range(len(locations)):
    ax.plot(lons[i], lats[i], color='blue', marker='o',
            markersize=6)
    ax.text(lons[i], lats[i]-0.5, locations[i],
            horizontalalignment='center', color='black',
            weight='extra bold', fontsize=12)
```

The `for` loop loops over all elements of the `range` defined by the length of `locations`. Since the length of `locations` is three, the range over which the `for` loop operates is from 0 to 2 by 1 – i.e., 0, 1, 2. It stores the appropriate number in the variable `i`, which can then be referred to in the looping code block to get data out of the `locations`, `lats`, and `lons` arrays.

The looping code block itself is indented four spaces, consistent with Python syntax. In it, there are two lines, the first of which plots the desired marker and the second of which plots the desired text label below the marker. Note how the first two arguments to each function are the location for the marker or text. The `text` function has the string to plot as its third argument. The remaining keywords passed to each function control how the marker and text are plotted: color, size, etc.

Go ahead and add the code defining the three lists and the looping code to your script and run it. You will obtain the same map, except the NWS offices located in Wisconsin will be labeled.

For this assignment, please create a map of the central United States (25-50°N, 87-105°W). Use the land_alt1 for the land cfeature.COLORS entry. Add gold-filled stars (marker type of '*') with a marker size of 15 and a 1-pixel-thick black line surrounding the star (with markeredgecolor and markeredgewidth keywords) at NWS radiosonde locations in the central United States. Add a bold black text label below each marker with the site's ID. Title your figure with your name, save it to a file, and turn in both your code and image by the start of the next class.

A list of all NWS radiosonde observation locations in the central United States is given on the next page. The first column is the station ID. The sixth and seventh columns are latitude and longitude, respectively, but each need to be divided by 10 (i.e., Aberdeen, SD is at 45.45°N, 98.41°W).

ABR	000076	ABERDEEN	SD	US	4545	-9841	0	0
AMA	000004	AMARILLO	TX	US	3523	-10170	0	0
BIS	000069	BISMARCK	ND	US	4677	-10075	0	0
BMX	000042	BIRMINGHAM	AL	US	3317	-8678	0	0
BRO	000087	BROWNSVILLE	TX	US	2591	-9741	0	0
CRP	000116	CORPUS_CHRISTI	TX	US	2777	-9750	0	0
DDC	000009	DODGE_CITY	KS	US	3776	-9996	0	0
DNR	000000	DENVER	CO	US	3975	-10487	0	0
DRT	000000	DEL_RIO	TX	US	2937	-10092	0	0
DVN	000084	QUAD_CITIES_IA	IL	US	4161	-9058	0	0
FWD	000041	DALLAS/FORT_WORTH	TX	US	3283	-9729	0	0
GRB	000051	GREEN_BAY	WI	US	4449	-8811	0	0
ILX	000114	LINCOLN	IL	US	4015	-8933	0	0
INL	000000	INTERNATIONAL_FALLS	MN	US	4857	-9340	0	0
JAN	000022	JACKSON	MS	US	3231	-9008	0	0
LBF	000096	NORTH_PLATTE	NE	US	4113	-10070	0	0
LCH	000120	LAKE_CHARLES	LA	US	3012	-9321	0	0
LIX	000046	NEW_ORLEANS	LA	US	3033	-8982	0	0
LZK	000020	LITTLE_ROCK	AR	US	3483	-9225	0	0
MAF	000079	MIDLAND/ODESSA	TX	US	3194	-10218	0	0
MPX	000088	TWIN_CITIES/CHANHASSEN	MN	US	4484	-9356	0	0
OAX	000055	OMAHA/VALLEY	NE	US	4132	-9636	0	0
OUN	000001	NORMAN	OK	US	3523	-9746	0	0
SGF	000077	SPRINGFIELD	MO	US	3723	-9340	0	0
SHV	000082	SHREVEPORT	LA	US	3245	-9384	0	0
TOP	000007	TOPEKA	KS	US	3907	-9563	0	0
RAP	000117	RAPID_CITY	SD	US	4407	-10321	0	0