Python programming — plotting

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

September 16, 2013

Numerical and scientific Python

matplotlib — Package with functions that resemble Matlab plotting

chaco — Package with more flexible and complex plotting than matplotlib

PIL — Python Imaging Library. Image processing

VTK — Visualization toolkit, 3D computer graphics and image processing

NetworkX — Network analysis and visualization

Mapnik — Geographical maps with Python bindings

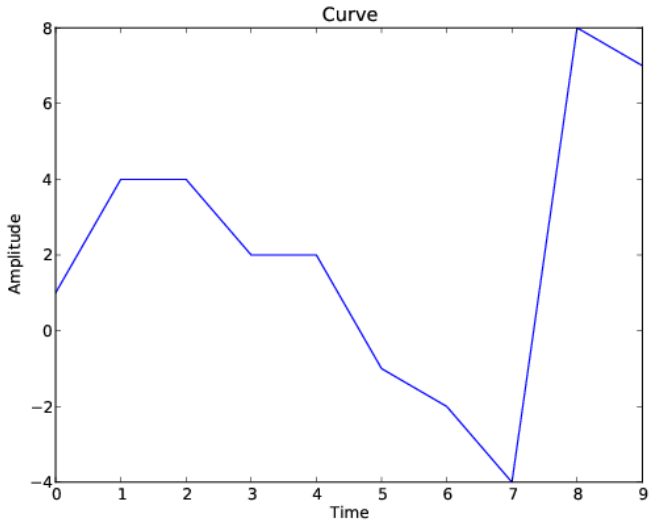
(**Gnuplot** — Gnuplot external program)

Matplotlib

Tutorial: http://matplotlib.sourceforge.net/users/pyplot_tutorial.html

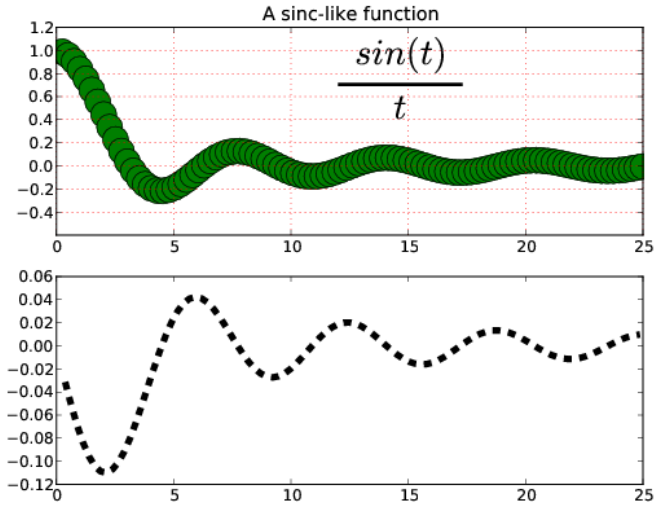
MATLAB commands in numerical Python (NumPy), A few examples in (Segaran, 2007, pp. 185+), (Bird et al., 2009, pp. 168–169)

Plot a curve in a window and also write an EPS (Encapsulated PostScript) file:




```
import matplotlib.pyplot as plt

plt.figure(1)
plt.plot([1, 4, 4, 2, 2, -1, -2, -4, 8, 7])
plt.xlabel("Time")
plt.ylabel("Amplitude")
plt.title("Curve")
plt.savefig("curve.eps")    # Could also have written svg, png, ...
plt.show()                  # Show figure in window
```

Matplotlib and NumPy




```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 25, 100)  # "t" is a "numpy.ndarray"
x = np.sin(t)/t              # "x" is also a "numpy.ndarray"
plt.subplot(2, 1, 1)
plt.plot(t, x, "go", markersize=20, alpha=0.5)
plt.title("A sinc-like function")
plt.text(12, 0.4, r"$\frac{sin(t)}{t}$", fontsize=40)     # LaTeX
plt.grid(color="r")
plt.axis((0, 25, -0.6, 1.2))
plt.subplot(2, 1, 2)
plt.plot(np.diff(t[:2])/2+t[:-1], np.diff(x), "k--", linewidth=5)
```

Example: Twitter geo heatmap . . .

Get geographical coordinates from Twitter and render them on a heatmap.

First the elaborate procedure for connecting to Twitter following and example: https://github.com/simplegeo/python-oauth2.

You initially need `comsumer_key` and `consumer_secret` and later the `oauth_verifier` number.

```
import oauth2 as oauth
import urlparse
import webbrowser

request_token_url = "https://twitter.com/oauth/request_token"
access_token_url = "https://twitter.com/oauth/access_token"
authorize_url = "https://twitter.com/oauth/authorize"
```

```
# This require you to setup an application on the Twitter homepage
consumer = oauth.Consumer(key=consumer_key, secret=consumer_secret)

client = oauth.Client(consumer)
response, content = client.request(request_token_url, "GET")
request_token = dict(urlparse.parse_qsl(content))

url = "%s?oauth_token=%s" % (authorize_url, request_token["oauth_token"])
webbrowser.open(url)

token = oauth.Token(request_token["oauth_token"],
                    request_token["oauth_token_secret"])
token.set_verifier(oauth_verifier)
client = oauth.Client(consumer, token)

response, content = client.request(access_token_url, "POST")
```

```
access_token = dict(urlparse.parse_qsl(content))

token = oauth.Token(access_token["oauth_token"],
                    access_token["oauth_token_secret"])
client = oauth.Client(consumer, token)
```

With the resulting `client` object you can use the `request` method to query the Twitter API, e.g.,

```
url = "https://api.twitter.com/1.1/search/tweets.json?q=geotag&rpp=100"
response = client.request(url)
```

Lets use that for the heatmap

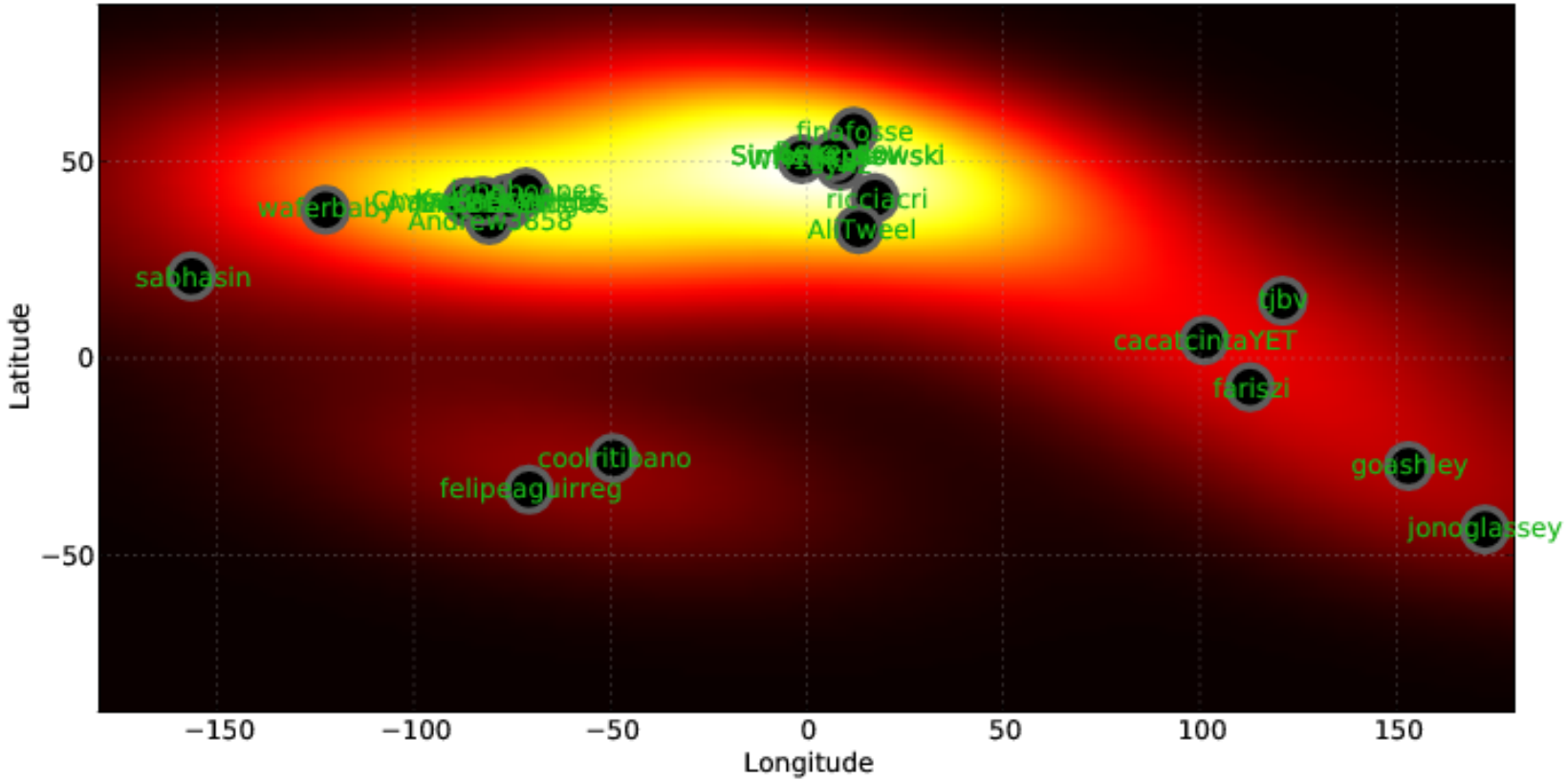
```
import simplejson as json
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm                    # Colormap
import scipy.stats                           # For kernel density estimation

# Get geo-tagged posts from Twitter
url = "https://api.twitter.com/1.1/search/tweets.json?q=geotag&count=100"
response, content = client.request(url)
tweets = json.loads(content)["statuses"]
coords = np.asarray([ t["geo"]["coordinates"][::-1]
                      for t in tweets if t["geo"] ])
users = [ t["user"]["screen_name"] for t in tweets if t["geo"] ]
```

Now `coords` variable contains (longitude, latitude) for posts in NumPy array while `users` contains the Twitter user names.

. . . Twitter geo heatmap

Construct a heatmap (http://www.scipy.org/SciPyPackages/Stats)

```
X, Y = np.mgrid[-180:180:100j, -90:90:100j]
positions = np.c_[X.ravel(), Y.ravel()]
kernel = scipy.stats.kde.gaussian_kde(coords.T)
Z = np.reshape(kernel(positions.T), X.shape)

plt.figure(), plt.hold(True)
plt.imshow(np.rot90(Z), cmap=cm.hot, extent=(-180, 180, -90, 90))
for user, c in zip(users, coords):
   plt.plot(c[0], c[1], "ko", markeredgewidth=3, markersize=20,
                              markeredgecolor=(0.3, 0.3, 0.3))
   dummy = plt.text(c[0], c[1], user, color=(.1, .7, 0.1),
             horizontalalignment="center", verticalalignment="center")

plt.axis((-180, 180, -90, 90)); plt.grid(color=(0.5, 0.5, 0.5))
plt.xlabel("Longitude"); plt.ylabel("Latitude")
```


50
0
−50
Latitude
−150
−100
−50
0
50
100
150
Longitude
sabhasin
fjby
fariszi
goashley
jonoglassey
felipeaguirreg

Dates and matplotlib . . .

```
from matplotlib.finance import quotes_historical_yahoo
from matplotlib.dates import num2date
from matplotlib.pyplot import *
from datetime import date

quotes = quotes_historical_yahoo("NVO", date(2011, 1, 1), date.today())
dates = [ num2date(row[0]) for row in quotes ]
closeprice = [ row[1] for row in quotes ]
plot(dates, closeprice)
show()
```

See also my blog: *NumPy beginner's guide: Date formatting, stock quotes and Wikipedia sentiment analysis*.

. . . Dates and matplotlib

Novo Nordisk stock quotes: Notice how matplotlib handles date information: The x-axis automagically shows datetime type in the plotted `dates` object.

Matplotlib

Many more examples on:

http://matplotlib.sourceforge.net/examples/

Chaco

Library by Enthought for interactive and static plotting.

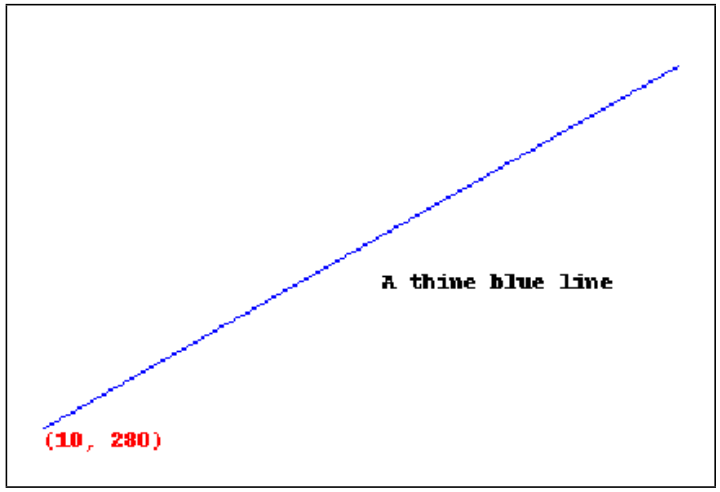
More elaborate than matplotlib, see quickstart for a "small" "hello, world" program with plotting of a line.

Script example from tutorial (slightly modified):

```
ipython --gui=wx
In [1]: from numpy import *
In [2]: from chaco.shell import *
In [3]: x = linspace(-2*pi, 2*pi, 100)
In [4]: plot(x, sin(x), "r-")
```

More information in tutorial.

PIL: Python Imaging Library



Creation, reading, writing and manipulation of image files (PNG, JPEG, PostScript, . . .), e.g., conversion, rotation, cropping, image sequence, filtering.

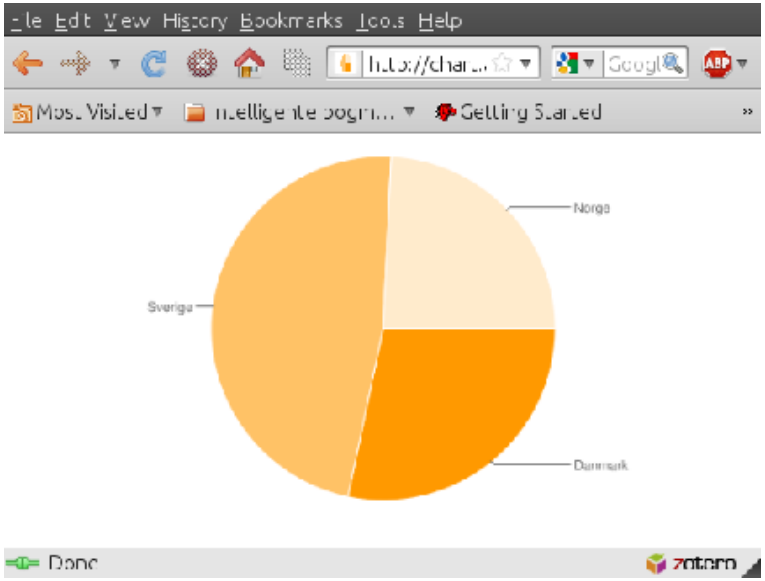
With `from PIL import Image, ImageDraw`, see also example (Segaran, 2007, p. 38+):

```
img = Image.new("RGB", (300, 200), (255, 255, 255))
draw = ImageDraw.Draw(img)
draw.line((10, 180, 290, 20), fill=(0, 0, 255))
draw.text((10, 180), "(10, 280)", (255, 0, 0))
draw.text((160, 110), "A thine blue line", (0, 0, 0))
img.save("thineblueline.png", "PNG")
```

Plotting via Google chart API

A data file `data.txt`:

```
5564219 Danmark
9325429 Sverige
4932400 Norge
```



Read in the data file, compute the sum and generate a URL so Google chart API will render a pie chart:

```
import urllib
d = open("data.txt").read().split()
s = reduce(lambda x, y: float(x)+float(y), d[0::2])       # The sum
url = "http://chart.apis.google.com/chart?cht=p&chd=t:" + \
      ",".join(map(lambda x : str(int(float(x)/s*100)),d[0::2])) + \
                    "&chs=600x300&chl=" + "|".join(d[1::2])
```

Google chart API has several other chart types.

VTK: 3D rendering

Modified "Step1" (`Cone1.py`) from VTK examples (Schroeder et al., 2006):

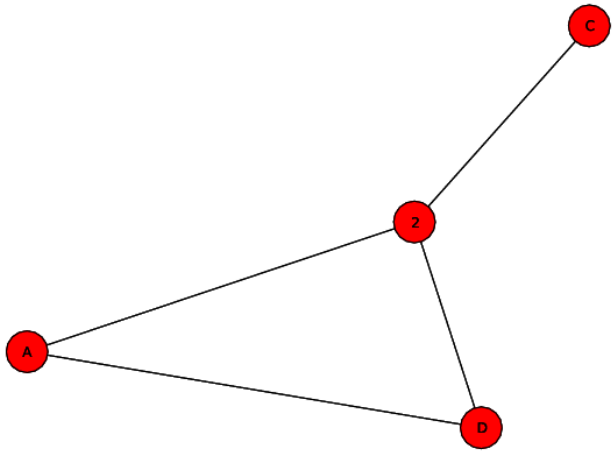
```
import vtk, time
cone = vtk.vtkConeSource()                          # Cone graphical object
coneMapper = vtk.vtkPolyDataMapper()                # "Mapper"
coneMapper.SetInputConnection(cone.GetOutputPort())
coneActor = vtk.vtkActor()                          # "Actor"
coneActor.SetMapper(coneMapper)
ren1= vtk.vtkRenderer()                             # "Renderer"
ren1.AddActor(coneActor)
renWin = vtk.vtkRenderWindow()                      # "Render Window"
renWin.AddRenderer(ren1)
while True:
    time.sleep(0.03)
    renWin.Render()
    ren1.GetActiveCamera().Azimuth(1)
```

NetworkX

Storing, analysis and visualization of graphs.

Documentation http://networkx.lanl.gov/

Examples in (Russell, 2011), small example in (Bird et al., 2009, pp. 169–170)



Simple undirected graph with layout via the *graphviz* program:

```
import networkx as nx                    # "nx" is the usual abbreviation
G = nx.Graph()
G.add_node("A")                          # Add single node
G.add_node(2)                            # Add another node (2)
G.add_nodes_from(["C", "D"])             # Add multiple nodes
G.add_edge("A", 2)                       # Add edge
G.add_edge(2, "C")                       # Add another edge
G.add_edges_from([("A", "D"), (2, "D")]) # Add multiple edges
nx.draw_graphviz(G, node_size=1000)      # or just nx.draw(G)
```

NetworkX functionalities

Graph types: Unidirected (`nx.Graph()`), directed graph (`nx.DiGraph()`), multigraphs that allows more than one edge between the same set of nodes (`nx.MultiGraph()` and `nx.MultiDiGraph()`). Weighted edges

IO and generations: Reading GML, edge list file, . . . , e.g., random graphs, "Karate club" data set, from NumPy matrix, . . .

Modifications: Add edges and nodes, graph set operations.

Analysis: Centrality, PageRank, shortest path distance, density, . . .

Visualizations: Drawing with different layouts. Uses graphviz.

Nodes can be any *hashable* object: string, integer, tuples, . . .

NetworkX example from NLTK book

```
import networkx as nx
import matplotlib.pyplot as plt
from nltk.corpus import wordnet as wn          # WordNet dictionary
```

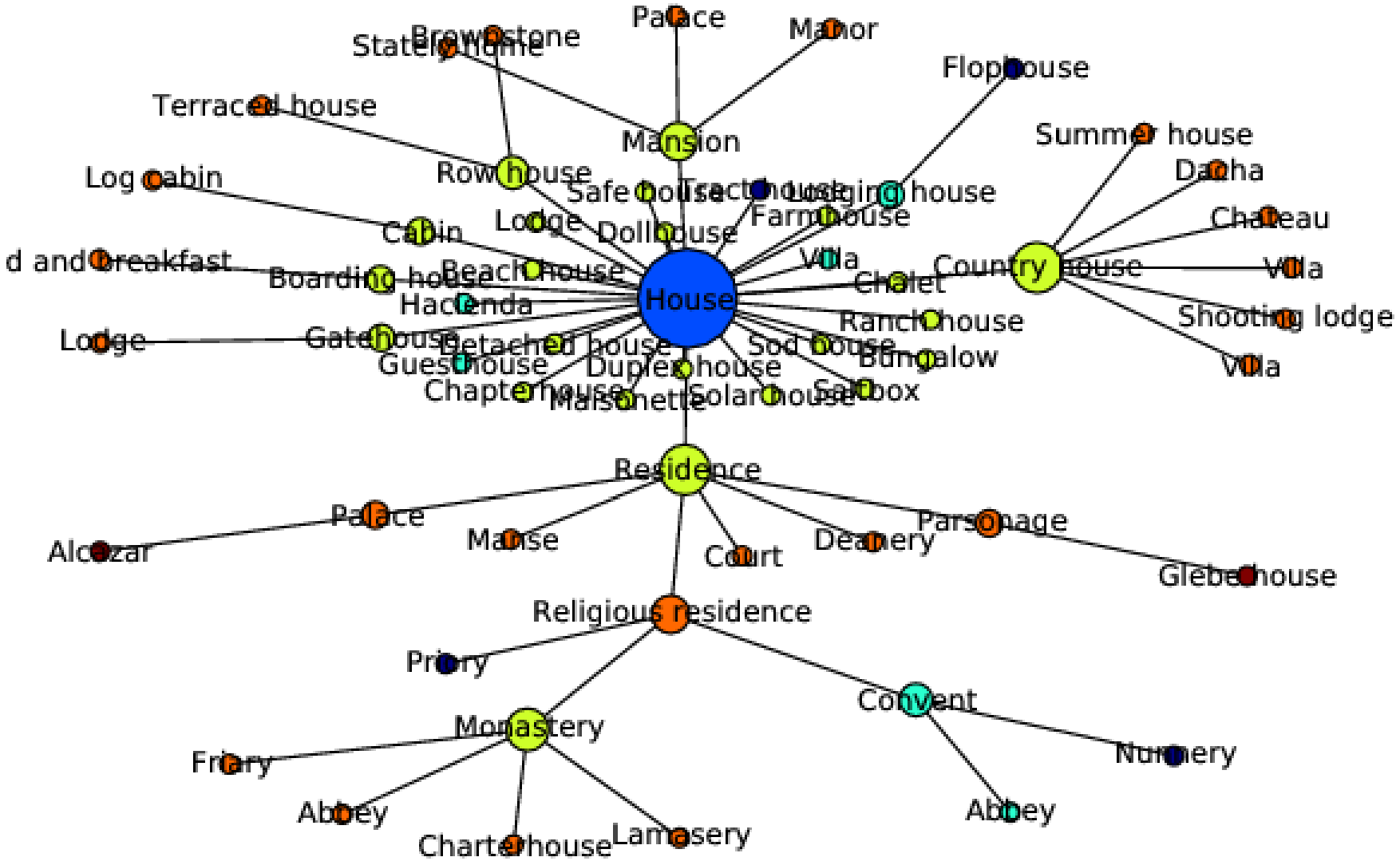
Get first meaning of the noun “dog” from WordNet via NLTK:

```
>>> dog = wn.synset("dog.n.01")
```

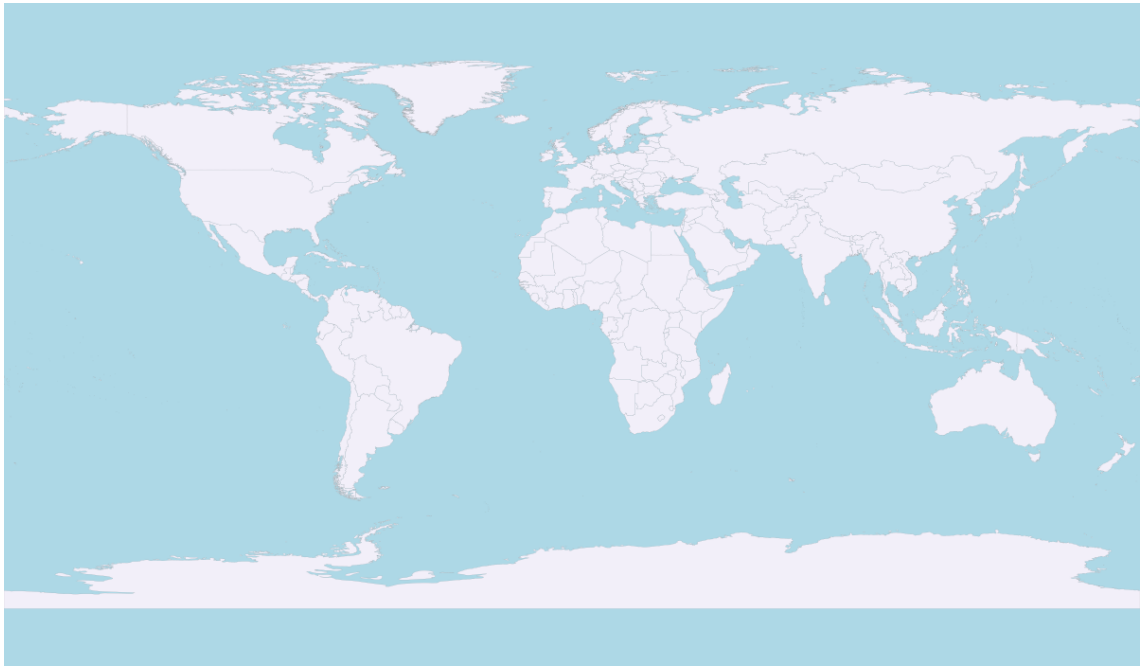
Show a few narrower senses of that meaning of “dog”:

```
>>> dog.hyponyms()[4:7]
[Synset('lapdog.n.01'), Synset('poodle.n.01'),
Synset('leonberg.n.01')]
>>> dog.hyponyms()[5].hyponyms()
[Synset('large_poodle.n.01'), Synset('miniature_poodle.n.01'),
Synset('toy_poodle.n.01'), Synset('standard_poodle.n.01')]
```

```
def traverse(graph, start, node=None):
    """Convert WordNet hyponym graph to NetworkX graph"""
    if not node: node = start
    graph.depth[node.name] = node.shortest_path_distance(start)
    for child in node.hyponyms():
        graph.add_edge(node.name, child.name)
        traverse(graph, start, child)

def nodename(name):
    """Convert, e.g., "shooting_lodge.n.01" to "Shooting lodge" """
    return re.sub("_", " ", re.sub("\..*", "", name)).capitalize()

G = nx.Graph();  G.depth = {}
traverse(G, wn.synset("house.n.01"))          # From the word "house"

nx.draw_graphviz(G, node_size=[50*G.degree(node) for node in G],
                    node_color=[G.depth[node] for node in G],
                    labels=dict([(g, nodename(g)) for g in G]))
plt.savefig("graph.png")
```


Palace
Manor
Brownstone
Flophouse
Terraced house
Mansion
Summer house
Log cabin
Row house
Dacha
Chateau
Cabin
Villa
Country house
Villa
House
Villa
Shooting lodge
Lodge
Villa
Residence
Palace
Alcazar
Manse
Court
Deanery
Parsonage
Glebe house
Religious residence
Priory
Convent
Monastery
Friary
Abbey
Abbey
Charterhouse
Lamasery

Mapnik



Package to construct maps.

Is used with OpenStreetMap

So-called styles may be setup in XML files or written in Python, e.g., to determine which color a “highway” should have.

Geographical data must be read from so-called data-sources.

Mapnik(2)

Example from Tutorial 1 – Getting started in Python on Mapnik's homepage:

```
import mapnik2 as mapnik

m = mapnik.Map(1200, 700, "+proj=latlong +datum=WGS84")
m.background = mapnik.Color("lightblue")

s = mapnik.Style()
r = mapnik.Rule()
r.symbols.append(mapnik.PolygonSymbolizer(mapnik.Color("#f2eff9")))
r.symbols.append(mapnik.LineSymbolizer(mapnik.Color("rgb(50%,50%,50%)"), 0.1))
s.rules.append(r)
m.append_style("My Style", s)

lyr = mapnik.Layer("world", "+proj=latlong +datum=WGS84")
lyr.datasource = mapnik.Shapefile(file="ne_110m_admin_0_countries.shp")
lyr.styles.append("My Style")
m.layers.append(lyr)

m.zoom_to_box(lyr.envelope())
mapnik.render_to_file(m, "world.png", "png")
```

Note required data for the example:

```
import urllib, zipfile
url = "https://github.com/mapnik/mapnik/wiki/data/110m-admin-0-countries.zip"
zipfile.ZipFile(urllib.urlretrieve(url)[0]).extractall()
```

References

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly, Sebastopol, California. ISBN 9780596516499.

Russell, M. A. (2011). *Mining the Social Web*. O'Reilly. ISBN 978-1-4493-8834-8.

Schroeder, W., Martin, K., and Lorensen, B. (2006). *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware. ISBN 1-930934-19-X.

Segaran, T. (2007). *Programming Collective Intelligence*. O'Reilly, Sebastopol, California.