Chapter 21. Py at Work

> *"Business!" cried the Ghost, wringing its hands again. "Mankind was my business..."*
>
> —Charles Dickens, A Christmas Carol

The businessman's uniform is a suit and tie. But before he can *get down to business*, he tosses his jacket over a chair, loosens his tie, rolls up his sleeves, and pours some coffee. Meanwhile, the business woman has already been getting work done. Maybe with a latte.

In business and government, we use all of the technologies from the earlier chapters—databases, the web, systems, and networks. Python's productivity is making it more popular in the enterprise and with startups.

Organizations have long fought incompatible file formats, arcane network protocols, language lock-in, and the universal lack of accurate documentation. They can create faster, cheaper, stretchier applications by using with tools such as these:

- Dynamic languages like Python
- The web as a universal graphical user interface
- RESTful APIs as language-independent service interfaces
- Relational and NoSQL databases
- "Big data" and analytics
- Clouds for deployment and capital savings

The Microsoft Office Suite

Business is heavily dependent on Microsoft Office applications and file formats. Although they are not well known, and in some cases poorly documented, there are some Python libraries that can help. Here are some that process Microsoft Office documents:

docx

This library creates, reads, and writes Microsoft Office Word 2007 *.docx* files.

python-excel

This one discusses the `xlrd`, `xlwt`, and `xlutils` modules via a PDF tutorial. Excel can also read and write comma-separated values (CSV) files, which you know how to process by using the standard `csv` module.

oletools

This library extracts data from Office formats.

OpenOffice is an open source alternative to Office. It runs on Linux, Unix, Windows, and macOS, and reads and writes Office file formats, It also installs a version of Python 3 for its own use. You can program OpenOffice in Python with the PyUNO library.

OpenOffice was owned by Sun Microsystems, and when Oracle acquired Sun, some people feared for its future availability. LibreOffice was spun off as a result. DocumentHacker describes using the Python UNO library with LibreOffice.

OpenOffice and LibreOffice had to reverse engineer the Microsoft file formats, which is not easy. The Universal Office Converter module depends on the UNO library in OpenOffice or LibreOffice. It can convert many file formats: documents, spreadsheets, graphics, and presentations.

If you have a mystery file, python-magic can guess its format by analyzing specific byte sequences.

The python open document library lets you provide Python code within templates to create dynamic documents.

Although not a Microsoft format, Adobe's PDF is very common in business. ReportLab has open source and commercial versions of its Python-based PDF generator. If you need to edit a PDF, you might find some help at StackOverflow.

Carrying Out Business Tasks

You can find a Python module for almost anything. Visit PyPI and type something into the search box. Many modules are interfaces to the public APIs of various services. You might be interested in some examples related to business tasks:

- Ship via Fedex or UPS.
- Mail with the stamps.com API.
- Read a discussion of *Python for business intelligence*.
- If Aeropresses are flying off the shelves in Anoka, was it customer activity or poltergeists? Cubes is an Online Analytical Processing (OLAP) web server and data browser.
- OpenERP is a large commercial Enterprise Resource Planning (ERP) system written in Python and JavaScript, with thousands of add-on modules.

Processing Business Data

Businesses have a particular fondness for data. Sadly, many of them conjure up perverse ways of making data harder to use.

Spreadsheets were a good invention, and over time businesses became addicted to them. Many nonprogrammers were tricked into programming because they were called *macros* instead of programs. But the universe is expanding and data is trying to keep up. Older versions of Excel were limited to 65,536 rows, and even newer versions choke at a million or so. When an organization's data outgrow the limits of a single computer, it's like headcount growing past a hundred people or so—suddenly you need new layers, intermediaries, and communication.

Excessive data programs aren't caused by the size of data on single desktops; rather, they're the result of the aggregate of data pouring into the business. Relational databases handle millions of rows without exploding, but only so many writes or updates at a time. A plain old text or binary file can grow gigabytes in size, but if you need to process it all at once,

you need enough memory. Traditional desktop software isn't designed for all this. Companies such as Google and Amazon had to invent solutions to handle so much data at scale. Netflix is an example built on Amazon's AWS cloud, using Python to glue together RESTful APIs, security, deployment, and databases.

Extracting, Transforming, and Loading

The underwater portions of the data icebergs include all the work to get the data in the first place. If you speak enterprise, the common term is extract, transform, load, or *ETL*. Synonyms such as *data munging* or *data wrangling* give the impression of taming an unruly beast, which might be apt metaphors. This would seem to be a solved engineering matter by now, but it remains largely an art. I talked a bit about this in Chapter 12. We address *data science* more broadly in Chapter 22, because this is where most developers spend a large part of their time.

If you've seen *The Wizard of Oz*, you probably remember (besides the flying monkeys) the part at the end—when the good witch told Dorothy that she could always go home to Kansas just by clicking her ruby slippers. Even when I was young I thought, "Now she tells her!" Although, in retrospect, I realize the movie would have been much shorter if she'd shared that tip earlier.

But this isn't a movie; we're talking about the world of business here, where making tasks shorter is a good thing. So, let me share some tips with you now. Most of the tools that you need for day-to-day data work in business are those that you've already read about here. Those include high-level data structures such as dictionaries and objects, thousands of standard and third-party libraries, and an expert community that's just a google away.

If you're a computer programmer working for some business, your workflow almost always includes the following:

1. Extracting data from weird file formats or databases
2. "Cleaning up" the data, which covers a lot of ground, all strewn with pointy objects

3. Converting things like dates, times, and character sets
4. Actually doing something with the data
5. Storing resulting data in a file or database
6. Rolling back to step 1 again; lather, rinse, repeat

Here's an example: you want to move data from a spreadsheet to a database. You can save the spreadsheet in CSV format and use the Python libraries from Chapter 16. Or, you can look for a module that reads the binary spreadsheet format directly. Your fingers know how to type `python excel` into Google and find sites such as Working with Excel files in Python. You can install one of the packages by using `pip`, and locate a Python database driver for the last part of the task. I mentioned SQLAlchemy and the direct low-level database drivers in that same chapter. Now you need some code in the middle, and that's where Python's data structures and libraries can save you time.

Let's try an example here, and then we'll try again with a library that saves a few steps. We'll read a CSV file, aggregate the counts in one column by unique values in another, and print the results. If we did this in SQL, we would use SELECT, JOIN, and GROUP BY.

First, the file, *zoo.csv*, which has columns for the type of animal, how many times it has bitten a visitor, the number of stitches required, and how much we've paid the visitor not to tell local television stations:

```
animal,bites,stitches,hush
bear,1,35,300
marmoset,1,2,250
bear,2,42,500
elk,1,30,100
weasel,4,7,50
duck,2,0,10
```

We want to see which animal is costing us the most, so we aggregate the total hush money by the type of animal. (We'll leave bites and stitches to an intern.) We use the `csv` module from "CSV" and `Counter` from "Count Items with Counter()". Save this code as *zoo_counts.py*:

```
import csv
from collections import Counter

counts = Counter()
with open('zoo.csv', 'rt') as fin:
    cin = csv.reader(fin)
    for num, row in enumerate(cin):
        if num > 0:
            counts[row[0]] += int(row[-1])
for animal, hush in counts.items():
    print("%10s %10s" % (animal, hush))
```

We skipped the first row because it contained only the column names. `counts` is a `Counter` object, and takes care of initializing the sum for each animal to zero. We also applied a little formatting to right-align the output. Let's try it:

```
$ python zoo_counts.py
      duck         10
       elk        100
      bear        800
    weasel         50
  marmoset        250
```

Hah! It was the bear. He was our prime suspect all along, but now we have the numbers.

Next, let's replicate this with a data processing toolkit called Bubbles. You can install it by typing this command:

```
$ pip install bubbles
```

It requires SQLAlchemy; if you don't have that, `pip install sqlalchemy` will do the trick. Here's the test program (call it *bubbles1.py*), adapted from the documentation:

```
import bubbles
```

```
p = bubbles.Pipeline()
p.source(bubbles.data_object('csv_source', 'zoo.csv', infer_fields=True))
p.aggregate('animal', 'hush')
p.pretty_print()
```

And now, the moment of truth:

```
$ python bubbles1.py
2014-03-11 19:46:36,806 DEBUG calling aggregate(rows)
2014-03-11 19:46:36,807 INFO called aggregate(rows)
2014-03-11 19:46:36,807 DEBUG calling pretty_print(records)
+--------+--------+------------+
|animal  |hush_sum|record_count|
+--------+--------+------------+
|duck    |      10|           1|
|weasel  |      50|           1|
|bear    |     800|           2|
|elk     |     100|           1|
|marmoset|     250|           1|
+--------+--------+------------+
2014-03-11 19:46:36,807 INFO called pretty_print(records)
```

If you read the documentation, you can avoid those debug print lines, and maybe change the format of the table.

Looking at the two examples, we see that the `bubbles` example used a single function call (`aggregate`) to replace our manual reading and counting of the CSV format. Depending on your needs, data toolkits can save a lot of work.

In a more realistic example, our zoo file might have thousands of rows (it's a dangerous place), with misspellings such as `bare` , commas in numbers, and so on. For good examples of practical data problems with Python code, I'd also recommend the following:

- *Data Crunching: Solve Everyday Problems Using Java, Python, and More*—Greg Wilson (Pragmatic Bookshelf).
- *Automate the Boring Stuff*—Al Sweigart (No Starch).

Data cleanup tools can save a lot of time, and Python has many of them. For another example, PETL does row and column extraction and renaming. Its related work page lists many useful modules and products. Chapter 22 has detailed discussions of some especially useful data tools: Pandas, NumPy, and IPython. Although they're currently best known among scientists, they're becoming popular among financial and data developers. At the 2012 Pydata conference, AppData discussed how these three and other Python tools help process 15 terabytes of data daily. Python handles very large real-world data loads.

You may also look back at the data serialization and validation tools discussed in "Data Serialization".

Data Validation

When cleaning up data, you'll often need to check:

- Data type, such as integer, float, or string
- Range of values
- Correct values, such as a working phone number or email address
- Duplicates
- Missing data

This is especially common when processing web requests and responses.

Useful Python packages for particular data types include:

- `validate_email`
- `phonenumber`

Some useful general tools are:

- `validators`
- `pydantic` —For Python 3.6 and above; uses type hints
- `marshmallow` —Also serializes and deserializes
- `cerberus`
- many others

Additional Sources of Information

Sometimes, you need data that originates somewhere else. Some business and government data sources include:

data.gov
: A gateway to thousands of data sets and tools. Its APIs are built on CKAN, a Python data management system.

Opening government with Python
: See the video and slides.

python-sunlight
: Libraries to access the Sunlight APIs.

froide
: A Django-based platform for managing freedom of information requests.

30 places to find open data on the web
: Some handy links.

Open Source Python Business Packages

Odoo
: Extensive ERP platform

Tryton
: Another extensive business platform

Oscar
: Ecommerce framework for Django

Grid Studio
: Python-based spreadsheet, runs locally or in the cloud

Python in Finance

Recently, the financial industry has developed a great interest in Python. Adapting software from Chapter 22 as well as some of their own, *quants* are building a new generation of financial tools:

Quantitative economics
: A tool for economic modeling, with lots of math and Python code

Python for finance
: Features the book *Derivatives Analytics with Python: Data Analytics, Models, Simulation, Calibration, and Hedging* by Yves Hilpisch (Wiley)

Quantopian
: An interactive website on which you can write your own Python code and run it against historic stock data to see how it would have done

PyAlgoTrade
: Another that you can use for stock backtesting, but on your own computer

Quandl
: Search millions of financial datasets

Ultra-finance
: A real-time stock collection library

Python for Finance *(O'Reilly)*
: A book by Yves Hilpisch with Python examples for financial modeling

Business Data Security

Security is a special concern for business. Entire books are devoted to this

topic, so we just mention a few Python-related tips here.

- "Scapy" discusses `scapy`, a Python-powered language for packet forensics. It has been used to explain some major network attacks.
- The Python Security site has discussions of security topics, details on some Python modules, and cheat sheets.
- The book *Violent Python* (subtitled *A Cookbook for Hackers, Forensic Analysts, Penetration Testers and Security Engineers*) by TJ O'Connor (Syngress) is an extensive review of Python and computer security.

Maps

Maps have become valuable to many businesses. Python is very good at making maps, so we're going to spend a little more time in this area. Managers love graphics, and if you can quickly whip up a nice map for your organization's website it wouldn't hurt.

In the early days of the web, I used to visit an experimental mapmaking website at Xerox. When big sites such as Google Maps came along, they were a revelation (along the lines of "why didn't I think of that and make millions?"). Now mapping and *location-based services* are everywhere, and are particularly useful in mobile devices.

Many terms overlap here: mapping, cartography, GIS (geographic information system), GPS (Global Positioning System), geospatial analysis, and many more. The blog at Geospatial Python has an image of the "800-pound gorilla" systems—GDAL/OGR, GEOS, and PROJ.4 (projections)—and surrounding systems, represented as monkeys. Many of these have Python interfaces. Let's talk about some of these, beginning with the simplest formats.

Formats

The mapping world has lots of formats: vector (lines), raster (images), metadata (words), and various combinations.

Esri, a pioneer of geographic systems, invented the *shapefile* format over

20 years ago. A shapefile actually consists of multiple files, including at the very least the following:

.shp
: The "shape" (vector) information

.shx
: The shape index

.dbf
: An attribute database

Let's grab a shapefile for our next example—visit the Natural Earth 1:110m Cultural Vectors page. Under "Admin 1 - States and Provinces," click the green download states and provinces box to download a zip file. After it downloads to your computer, unzip it; you should see these resulting files:

```
ne_110m_admin_1_states_provinces_shp.README.html
ne_110m_admin_1_states_provinces_shp.sbn
ne_110m_admin_1_states_provinces_shp.VERSION.txt
ne_110m_admin_1_states_provinces_shp.sbx
ne_110m_admin_1_states_provinces_shp.dbf
ne_110m_admin_1_states_provinces_shp.shp
ne_110m_admin_1_states_provinces_shp.prj
ne_110m_admin_1_states_provinces_shp.shx
```

We'll use these for our examples.

Draw a Map from a Shapefile

This section is an overly simplified demonstration of reading and displaying a shapefile. You'll see that the result has problems, and that you'd be better off working with a higher-level mapping package, such as those in the sections that follow.

You'll need this library to read a shapefile:

```
$ pip install pyshp
```

Now for the program, *map1.py*, which I've modified from a Geospatial Python blog post:

```
def display_shapefile(name, iwidth=500, iheight=500):
    import shapefile
    from PIL import Image, ImageDraw
    r = shapefile.Reader(name)
    mleft, mbottom, mright, mtop = r.bbox
    # map units
    mwidth = mright - mleft
    mheight = mtop - mbottom
    # scale map units to image units
    hscale = iwidth/mwidth
    vscale = iheight/mheight
    img = Image.new("RGB", (iwidth, iheight), "white")
    draw = ImageDraw.Draw(img)
    for shape in r.shapes():
        pixels = [
            (int(iwidth - ((mright - x) * hscale)), int((mtop - y) * vscale))
            for x, y in shape.points]
        if shape.shapeType == shapefile.POLYGON:
            draw.polygon(pixels, outline='black')
        elif shape.shapeType == shapefile.POLYLINE:
            draw.line(pixels, fill='black')
    img.show()

if __name__ == '__main__':
    import sys
    display_shapefile(sys.argv[1], 700, 700)
```

This reads the shapefile and iterates through its individual shapes. I'm checking for only two shape types: a polygon, which connects the last point to the first, and a polyline, which doesn't. I've based my logic on the original post and a quick look at the documentation for `pyshp`, so I'm not really sure how it will work. Sometimes, we just need to make a start and deal with any problems as we find them.

So, let's run it. The argument is the base name of the shapefile files, with-

out any extension:

```
$ python map1.py ne_110m_admin_1_states_provinces_shp
```

You should see something like Figure 21-1.

Well, it drew a map that resembles the United States, but:

- It looks like a cat dragged yarn across Alaska and Hawaii; this is a *bug*.
- The country is squished; I need a *projection*.
- The picture isn't pretty; I need better *style* control.

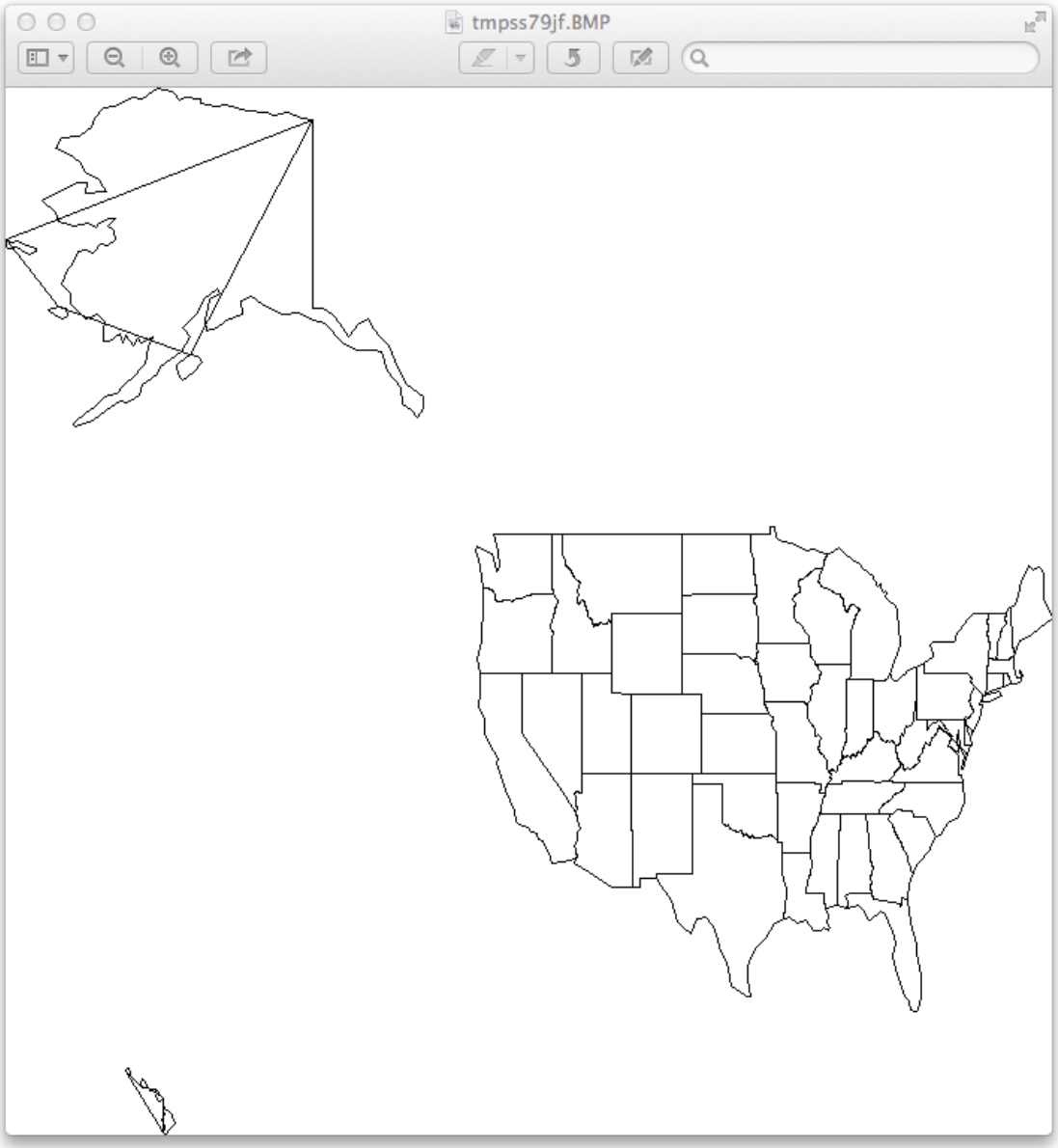

Figure 21-1. Preliminary map

To address the first point: I have a problem somewhere in my logic, but

what should I do? Chapter 19 discusses development tips, including debugging, but we can consider other options here. I could write some tests and bear down until I fix this, or I could just try some other mapping library. Maybe something at a higher level would solve all three of my problems (the stray lines, squished appearance, and primitive style).

As far as I can tell, there is no bare-bones pure Python mapping package. Luckily, there are some fancier ones, so let's take a look.

Geopandas

Geopandas integrates `matplotlib`, `pandas`, and other Python libraries into a geospatial data platform.

The base package is installed with the familiar `pip install geopandas`, but it relies on other packages that you will also need to install with `pip` if you don't have them already:

- `numpy`
- `pandas` (version 0.23.4 or later)
- `shapely` (interface to GEOS)
- `fiona` (interface to GDAL)
- `pyproj` (interface to PROJ)
- `six`

Geopandas can read shapefiles (including those from the previous section), and handily includes two from Natural Earth: country/continent outlines, and national capital cities. Example 21-1 is a simple demo that uses both of them.

Example 21-1. geopandas.py

```
import geopandas
import matplotlib.pyplot as plt

world_file = geopandas.datasets.get_path('naturalearth_lowres')
world = geopandas.read_file(world_file)
cities_file = geopandas.datasets.get_path('naturalearth_cities')
```

```
cities = geopandas.read_file(cities_file)
base = world.plot(color='orchid')
cities.plot(ax=base, color='black', markersize=2)
plt.show()
```

Run that, and you should get the map shown in Figure 21-2.

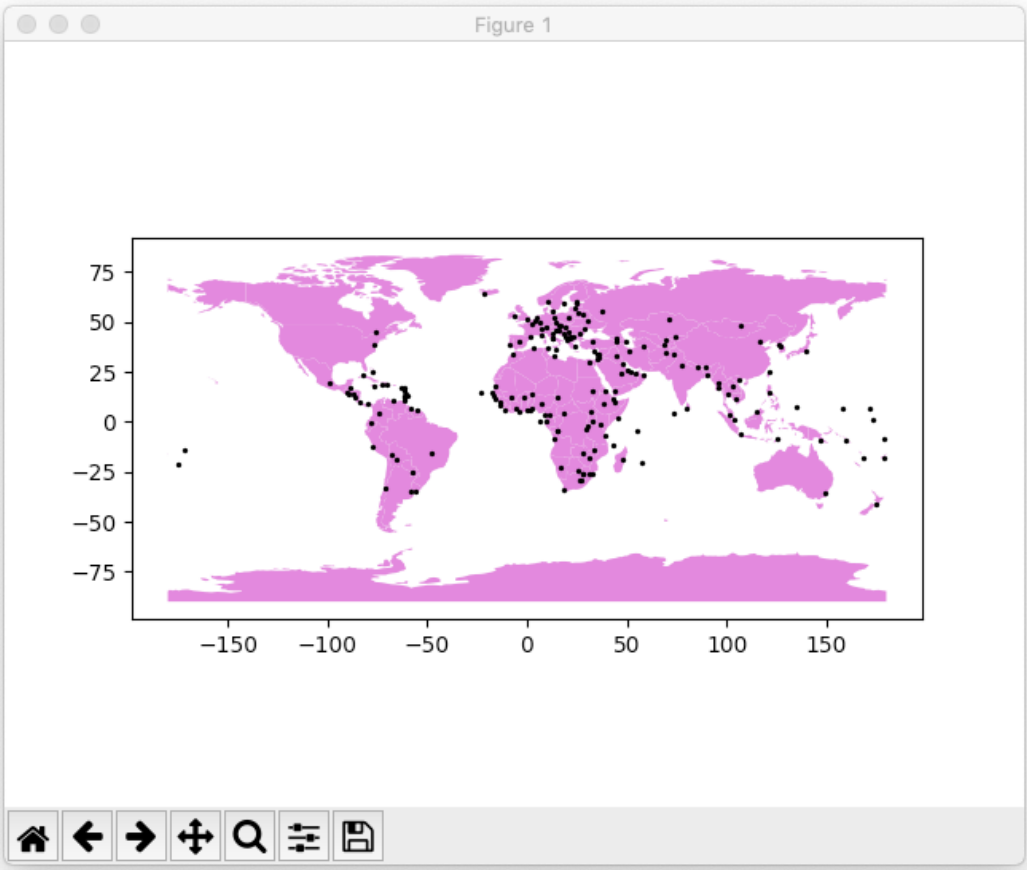

Figure 21-2. Geopandas map

To me, geopandas currently looks like the best combination for geographic data management and display. But there are many worthy contenders, which we look at in the next section.

Other Mapping Packages

Here's a grab bag of links to other Python mapping software; many cannot be installed fully with `pip`, but some can with `conda` (the alternative Python package installer that's especially handy for scientific software):

pyshp

A pure-Python shapefile library, mentioned earlier in "Draw a Map from a Shapefile".

kartograph

Renders shapefiles into SVG maps on the server or client.

shapely

Addresses geometric questions such as, "What buildings in this town are within the 50-year flood contour?"

basemap

Based on `matplotlib`, draws maps and data overlays. Unfortunately, it has been deprecated in favor of Cartopy.

cartopy

Succeeds Basemap, and does some of the things that `geopandas` does.

folium

Works with leaflet.js, used by `geopandas`.

plotly

Another plotting package that includes mapping features.

dash

Uses Plotly, Flask and JavaScript to create interactive visualizations, including maps.

fiona

Wraps the OGR library, which handles shapefiles and other vector formats.

Open Street Map

Accesses the vast OpenStreetMap world maps.

mapnik

A C++ library with Python bindings, for vector (line) and raster (image) maps.

Vincent

Translates to Vega, a JavaScript visualization tool; see the tutorial: Mapping data in Python with pandas and vincent.

Python for ArcGIS

Links to Python resources for Esri's commercial ArcGIS product.

Using geospatial data with python

Video presentations.

So you'd like to make a map using Python

Uses `pandas`, `matplotlib`, `shapely`, and other Python modules to create maps of historic plaque locations.

Python Geospatial Development *(Packt)*

A book by Eric Westra with examples using `mapnik` and other tools.

Learning Geospatial Analysis with Python *(Packt)*

Another book by Joel Lawhead reviewing formats and libraries, with geospatial algorithms.

geomancer

Geospatial engineering, such as the distance from a point to the nearest Irish bar.

If you're interested in maps, try downloading and installing one of these packages and see what you can do. Or, you can avoid installing software and try connecting to a remote web service API yourself; Chapter 18 shows you how to connect to web servers and decode JSON responses.

Applications and Data

I've been talking about drawing maps, but you can do a lot more with map data. *Geocoding* converts between addresses and geographic coordinates. There are many geocoding APIs (see ProgrammableWeb's comparison) and Python libraries:

- geopy
- pygeocoder
- googlemaps

If you sign up with Google or another source to get an API key, you can access other services such as step-by-step travel directions or local search.

Here are a few sources of mapping data:

http://www.census.gov/geo/maps-data
: Overview of the US Census Bureau's map files

http://www.census.gov/geo/maps-data/data/tiger.html
: Heaps of geographic and demographic map data

http://wiki.openstreetmap.org/wiki/Potential_Datasources
: Worldwide sources

http://www.naturalearthdata.com
: Vector and raster map data at three scales

I should mention the Data Science Toolkit here. It includes free bidirectional geocoding, coordinates to political boundaries and statistics, and more. You can also download all the data and software as a virtual machine (VM) and run it self-contained on your own computer.

Coming Up

We go to a Science Fair and see all the Python exhibits.

Things to Do

21.1 Install `geopandas` and run Example 21-1. Try modifying things like colors and marker sizes.