Extracted from:

Python Companion to Data Science

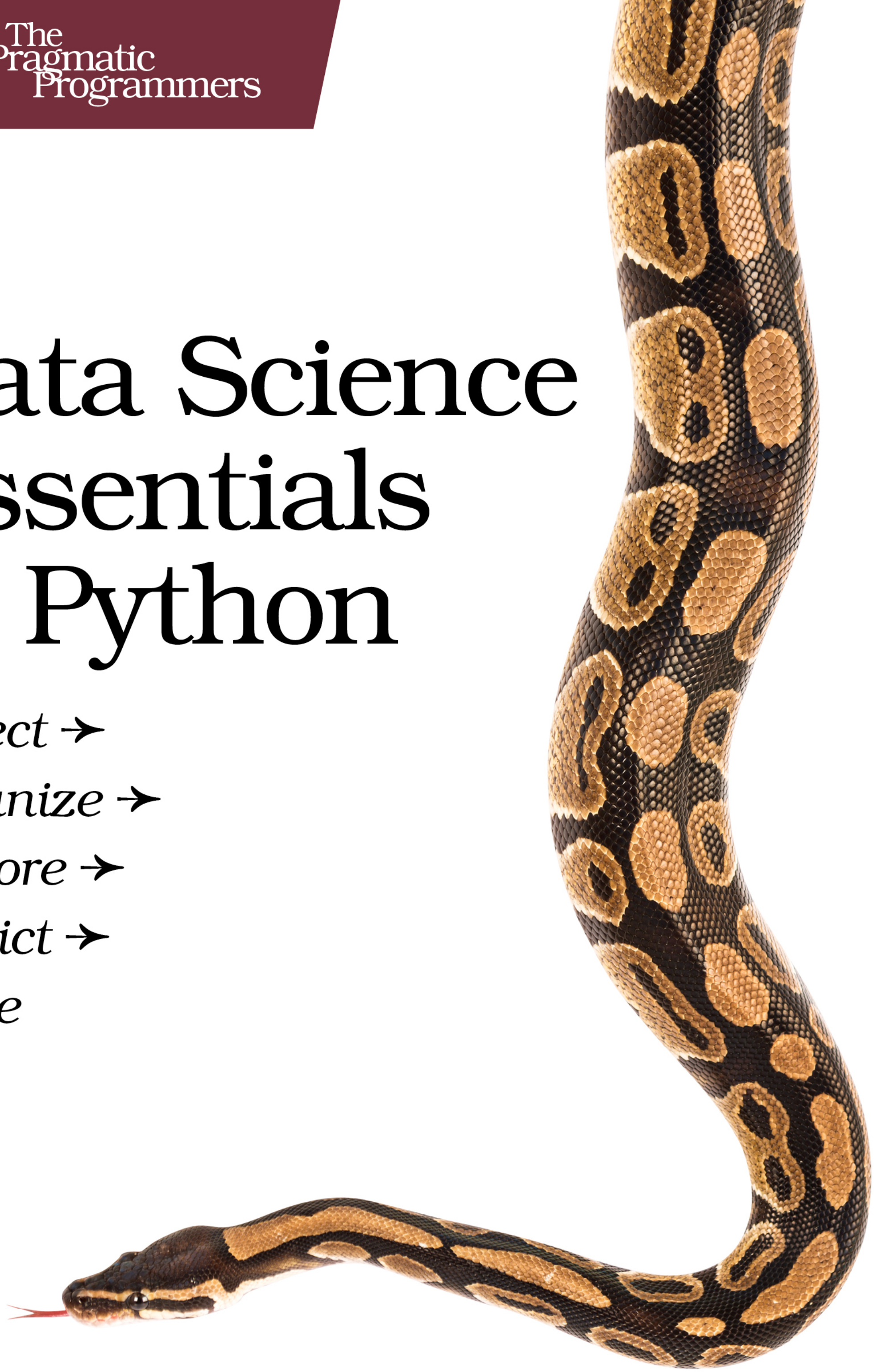
Collect → Organize → Explore → Predict → Value

The Pragmatic Bookshelf
Raleigh, North Carolina

The Pragmatic Programmers

Data Science Essentials in Python

Collect →
Organize →
Explore →
Predict →
Value



Dmitry Zinoviev

edited by Katharine Dvorak

Python Companion to Data Science

Collect → Organize → Explore → Predict → Value

Dmitry Zinoviev

The Pragmatic Bookshelf
Raleigh, North Carolina

Pragmatic Bookshelf

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Katharine Dvorak (editor)
Potomac Indexing, LLC (index)
Nicole Abramowitz (copyedit)
Gilson Graphics (layout)
Janet Furlow (producer)

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-68050-184-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—August 2016

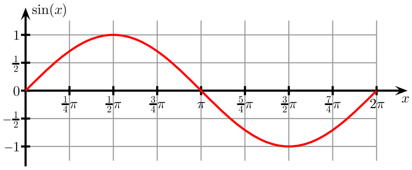
To my beautiful and most intelligent wife
Anna; to our children: graceful ballerina
Eugenia and romantic gamer Roman; and to
my first data science class of summer 2015.

> *"I am plotting for myself, and counterplotting the designs of others," replied Tresham, mysteriously.*
>
> ➤ *William Harrison Ainsworth, English historical novelist*

CHAPTER 8

Plotting

Plotting data is an essential part of any exploratory or predictive data analysis—and probably the most essential part of report writing. Frankly speaking, nobody wants to read reports without pictures, even if the pictures are irrelevant, like this elegant sine wave:



There are three principal approaches to programmable plotting. We start an *incremental* plot with a blank plot canvas and then add graphs, axes, labels, legends, and so on, incrementally using specialized functions. Finally, we show the plot image and optionally save it into a file. Examples of incremental plotting tools include the R language function plot(), the Python module pyplot, and the gnuplot command-line plotting program.

Monolithic plotting systems pass all necessary parameters, describing the graphs, charts, axes, labels, legends, and so on, to the plotting function. We plot, decorate, and save the final plot at once. An example of a monolithic plotting tool is the R language function xyplot().

Finally, *layered* tools represent what to plot, how to plot, and any additional features as virtual "layers"; we add more layers as needed to the "plot" object. An example of a layered plotting tool is the R language function ggplot(). (For the sake of aesthetic compatibility, the Python module matplotlib provides the ggplot plotting style.)

In this chapter, you'll take a look at how to do incremental plotting with pyplot.

Unit 41

Basic Plotting with PyPlot

Plotting for numpy and pandas is provided by the module matplotLib—namely, by the sub-module pyplot.

Let's start our experimentation with pyplot by invoking the spirit of the NIAAA surveillance report you converted into a frame earlier on page ?, and proceed to plotting alcohol consumption for different states and alcohol kinds over time. Unfortunately, as is always the case with incremental plotting systems, no single function does all of the plotting, so let's have a look at a complete example:

pyplot-images.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# The NIAAA frame has been pickled before
alco = pickle.load(open("alco.pickle", "rb"))
del alco["Total"]
columns, years = alco.unstack().columns.levels

# The state abbreviations come straight from the file
states = pd.read_csv(
    "states.csv",
    names=("State", "Standard", "Postal", "Capital"))
states.set_index("State", inplace=True)

# Alcohol consumption will be sorted by year 2009
frames = [pd.merge(alco[column].unstack(), states,
                   left_index=True, right_index=True).sort_values(2009)
          for column in columns]

# How many years are covered?
span = max(years) - min(years) + 1
```

The first code fragment simply imports all necessary modules and frames. It then combines NIAAA data and the state abbreviations into one frame and splits it into three separate frames by beverage type. The next code fragment is in charge of plotting.

pyplot-images.py

```
# Select a good-looking style
matplotlib.style.use("ggplot")

STEP = 5
# Plot each frame in a subplot
for pos, (draw, style, column, frame) in enumerate(zip(
        (plt.contourf, plt.contour, plt.imshow),    ❶
        (plt.cm.autumn, plt.cm.cool, plt.cm.spring),
        columns, frames)):

    # Select the subplot with 2 rows and 2 columns
    plt.subplot(2, 2, pos + 1)    ❷

    # Plot the frame
    draw(frame[frame.columns[:span]], cmap=style, aspect="auto")    ❸

    # Add embellishments
    plt.colorbar()    ❹
    plt.title(column)
    plt.xlabel("Year")
    plt.xticks(range(0, span, STEP), frame.columns[:span:STEP])
    plt.yticks(range(0, frame.shape[0], STEP), frame.Postal[::STEP])
    plt.xticks(rotation=-17)
```

The functions imshow(), contour(), and contourf() (at ❶) display the matrix as an image, a contour plot, and a filled contour plot, respectively. Don't use these three functions (or any other plotting functions) in the same subplot, because they superimpose new plots on the previously drawn plots—unless that's your intention, of course. The optional parameter cmap (at ❸) specifies a prebuilt palette (color map) for the plot.

You can pack several subplots of the same or different types into one master plot (at ❷). The function subplot(n, m, number) partitions the master plot into n virtual rows and m virtual columns and selects the subplot number. The subplots are numbered from 1, column-wise and then row-wise. (The upper-left subplot is 1, the next subplot to the right of it is 2, and so on.) All plotting commands affect only the most recently selected subplot.

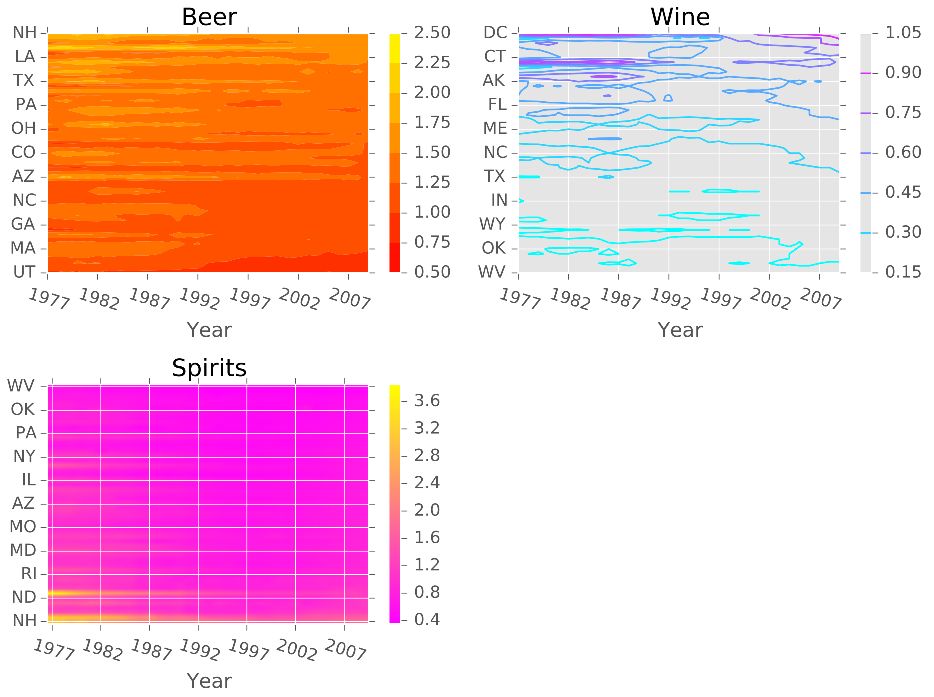
Note that the origin of the image plot is in the upper-left corner, and the Y axis goes down (that's how plotting is done in computer graphics), but the origin of all other plots is in the lower-left corner, and the Y axis goes up (that's how plotting is done in mathematics). Also, by default, an image plot and a contour plot of the same data have different aspect ratios, but you can make them look similar by passing the aspect="auto" option.

The functions colorbar(), title(), xlabel(), ylabel(), grid(), xticks(), yticks(), and tick_params() (at ❹) add the respective decorations to the plot. (We'll revisit them in Unit 43, *Mastering Embellishments*, on page 12.) The function grid() actually toggles the grid on and off, so whether you have a grid or not depends on whether you had it in the first place, which, in turn, is controlled by the plotting style.

The function tight_layout() adjusts subplots and makes them look nice and tight. Take a look at the following plots:

pyplot-images.py

```
plt.tight_layout()
plt.savefig("../images/pyplot-all.pdf")
#plt.show()
```



The function savefig() saves the current plot in a file. The function takes either a file name or an open file handle as the first parameter. If you pass the file name, savefig() tries to guess the image format from the file extension. The function supports many popular image file formats, but not GIF.

The function show() displays the plot on the screen. It also clears the canvas, but if you simply want to clear the canvas, call clf().

Unit 42

Getting to Know Other Plot Types

In addition to contour and image plots, pyplot supports a variety of more conventional plot types: bar plots, box plots, histograms, pie charts, line plots, log and log-log plots, scatter plots, polar plots, step plots, and so on. The online pyplot gallery[1] offers many examples, and the following table lists many of the pyplot plotting functions.

Plot type	Function
Vertical bar plot	bar()
Horizontal bar plot	barh()
Box plot with "whiskers"	boxplot()
Errorbar plot	errorbar()
Histogram (can be vertical or horizontal)	hist()
Log-log plot	loglog()
Log plot in X	semilogx()
Log plot in Y	semilogy()
Pie chart	pie()
Line plot	plot()
Date plot	plot_dates()
Polar plot	polar()
Scatter plot (size and color of dots can be controlled)	scatter()
Step plot	step()

Table 5—Some pyplot Plot Types

[1] matplotlib.org/gallery.html

Unit 43

Mastering Embellishments

With pyplot, you can control a lot of aspects of plotting.

You can set and change axes scales ("linear" vs. "log"—logarithmic) with the xscale(scale) and yscale(scale) functions, and you can set and change axes limits with the xlim(xmin, xmax) and ylim(ymin, ymax) functions.

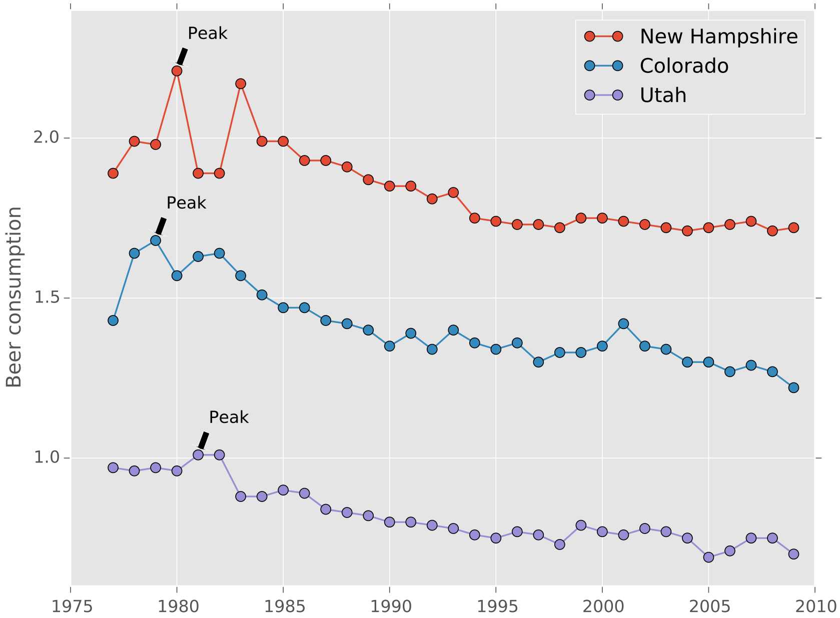
You can set and change font, graph, and background colors, and font and point sizes and styles.

You can also add notes with annotate(), arrows with arrow(), and a legend block with legend(). In general, refer to the pyplot documentation for the complete list of embellishment functions and their arguments, but let's at least add some arrows, notes, and a legend to an already familiar NIAAA graph:

pyplot-legend.py

```
import matplotlib, matplotlib.pyplot as plt
import pickle, pandas as pd

# The NIAAA frame has been pickled before
alco = pickle.load(open("alco.pickle", "rb"))

# Select the right data
BEVERAGE = "Beer"
years = alco.index.levels[1]
states = ("New Hampshire", "Colorado", "Utah")

# Select a good-looking style
plt.xkcd()
matplotlib.style.use("ggplot")

# Plot the charts
for state in states:
    ydata = alco.ix[state][BEVERAGE]
    plt.plot(years, ydata, "-o")
    # Add annotations with arrows
    plt.annotate(s="Peak", xy=(ydata.argmax(), ydata.max()),
                 xytext=(ydata.argmax() + 0.5, ydata.max() + 0.1),
                 arrowprops={"facecolor": "black", "shrink": 0.2})

# Add labels and legends
plt.ylabel(BEVERAGE + " consumption")
plt.title("And now in xkcd...")
plt.legend(states)

plt.savefig("../images/pyplot-legend-xkcd.pdf")
```

The triple-line plot shown here illustrates the dynamics of beer consumption in three states (in fact, in the most, least, and median beer-drinking states):

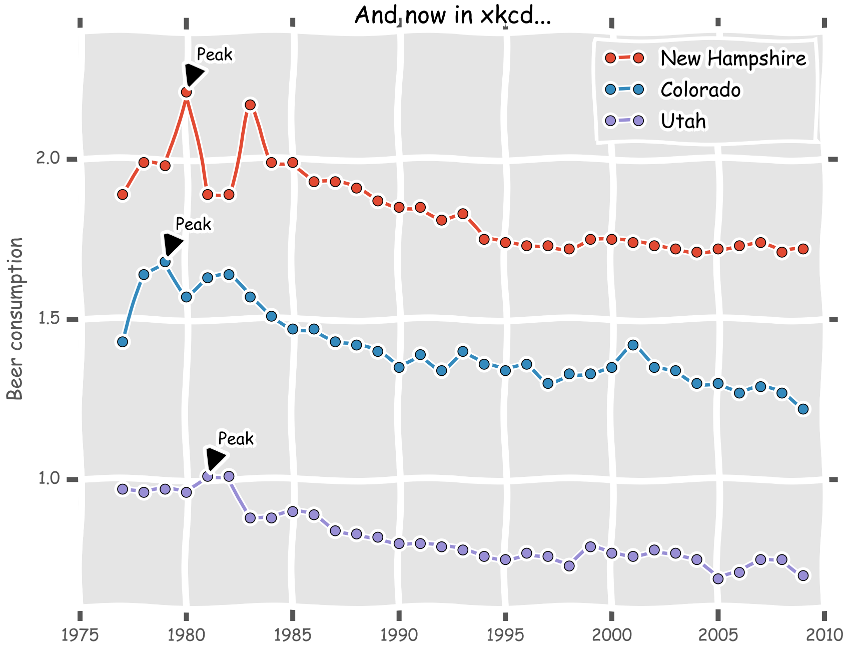


A Note on Unicode

If your plot contains Unicode (meaning non-Latin) characters, you may need to change the default font before plotting any text by adding a call to `matplotlib.rc("font", family="Arial")` as the first line of your plotting script.

Finally, you can change the style of a `pyplot` plot to resemble the popular xkcd[2] web comic with the function `xkcd()`. (The function affects only the plot elements added after the call to it.) For some reason, we can't save the plots as PostScript files, but everything else works. Nonetheless, you probably should avoid including xkcd-style plots in official presentations because they look as if drawn by a drunk (see the plot on page 14)—unless, of course, the presentation itself is about alcohol consumption.

[2] xkcd.com



The module pyplot is great on its own. But it's even better when combined with pandas, which we'll look at next.