Data Science from Scratch

First Principles with Python

Stephen CUI

March 20, 2023

Contents

Chapter 1

Getting Data

1.1 stdin and stdout

If you run your Python scripts at the command line, you can pipe data through them using `sys.stdin` and `sys.stdout`.

在Windows中的命令行可以输入：

```
type line_count.txt | python egrep.py "[0-9]"
type line_count.txt | python line_count.py
type line_count.txt | python egrep.py "[0-9]" | python line_count.py
```

The | is the pipe character, which means "use the output of the left command as the input of the right command." You can build pretty elaborate data-processing pipelines this way.

1.2 Reading Files

You can also explicitly read from and write to files directly in your code. Python makes working with files pretty simple.

1.2.1 The Basics of Text Files

The first step to working with a text file is to obtain a file object using `open`.

- 'r' means read-only, it's assumed if you leave it out
- 'w' is write – will destroy the file if it already exists!
- 'a' is append – for adding to the end of the file

Because it is easy to forget to close your files, you should always use them in a with block, at the end of which they will be closed automatically.

```
with open(filename) as f:
data = function_that_gets_data_from(f)

# at this point f has already been closed, so don't try to use it
process(data)
```

If you need to read a whole text file, you can just iterate over the lines of the file using `for`. Every line you get this way ends in a newline character, so you'll often want to `strip` it before doing anything with it.

1.2.2 Delimited Files

More frequently you'll work with files with lots of data on each line. These files are very often either *comma-separated* or *tab-separated*: each line has several fields, with a comma or a tab indicating where one field ends and the next field starts.

> Never parse a comma-separated file yourself. You will screw up the edge cases!

If your file has no headers (which means you probably want each row as a list, and which places the burden on you to know what's in each column), you can use `csv.reader` to iterate over the rows, each of which will be an appropriately split list.

If your file has headers, you can either skip the header row with an initial call to `reader.next`, or get each row as a dict (with the headers as keys) by using `csv.DictReader`. Even if your file doesn't have headers, you can still use DictReader by passing it the keys as a `fieldnames` parameter.

You can similarly write out delimited data using `csv.writer`. csv.writer will do the right thing if your fields themselves have commas in them.

1.3 Scraping the Web

Another way to get data is by scraping it from web pages. Fetching web pages, it turns out, is pretty easy; getting meaningful structured information out of them less so.(获取有意义的结构化数据就没有那么容易了)

1.3.1 HTML and the Parsing Thereof

To get data out of HTML, we will use the Beautiful Soup library, which builds a tree out of the various elements on a web page and provides a simple interface for accessing them. We'll also be using the Requests library, which is a much nicer way of making HTTP requests than anything that's built into Python.

Python's built-in HTML parser is not that lenient, which means that it doesn't always cope well with HTML that's not perfectly formed. For that reason, we'll also install the html5lib parser.

1.4 Using APIs

Many websites and web services provide *application programming interfaces* (APIs), which allow you to explicitly request data in a structured format.

1.4.1 JSON and XML

Because HTTP is a protocol for transferring text, the data you request through a web API needs to be serialized(串行化) into a string format. Often this serialization uses *JavaScript Object Notation* (JSON).

We can parse JSON using Python's `json` module. In particular, we will use its `loads` function, which deserializes a string representing a JSON object into a Python object.

Sometimes an API provider hates you and provides only responses in XML. You can use Beautiful Soup to get data from XML similarly to how we used it to get data from HTML.

1.4.2 Using an Unauthenticated API

Python doesn't come with a great date parser, so we'll need to install one:

```
python -m pip install python-dateutil
```

from which you'll probably only ever need the `dateutil.parser.parse` function.

1.4.3 Finding APIs

If you’re looking for a list of APIs that have Python wrappers, there’s a nice one from Real Python on GitHub.

1.5 Example: Using the Twitter APIs