Data Analysis

Colin Carroll

March 24, 2014

Contents

Chapter 1

Introduction

I've been working as an analytics developer for nearly two years now, and wanted to share the most helpful things I've learned in my transition from academia.

First, there are a dizzying number of tools and techniques to learn. The general framework I want to cover is:

1. Setting up and navigating a development environment from the command line (I use OSX and Ubuntu)
2. Scripting with python and R,
3. Saving and sharing work with git and github, and
4. Storing and accessing data via databases and sql

This will be opinionated, in that I'll typically present what I do as the right way to do things. Obviously there are other ways of doing things.

Chapter 2

Before You Start

2.1 Before You Start

We'll be moving mostly without a mouse. It is terrible at first, but pays dividends. If you are using Ubuntu you can get to a terminal with `ctrl+alt+T`. On OSX, use `cmd + space` to open spotlight, type in "terminal", then hit enter. I'll give some pointers to the most important commands to know in the terminal. Some important ones to start:

1. `$ ls` lists the files in the current directory
2. `$ cd directory` changes your directory to `directory` (if it exists).
3. The up/down keys scroll through recent commands
4. `tab` will autocomplete when it can. Sometimes you need to hit it twice and it will give you options
5. `ctrl+r` Gives you reverse search in the terminal, letting you type in a few letters. Hitting `ctrl+r` more scrolls backwards in these options, and `ctrl+g` cancel your search.

2.2 Setting up and navigating a development environment from the command line

If you can use an Ubuntu box, it will be much easier to set everything up, but it isn't terrible on OS X, either. In general, the better you can find your

question on Stack Overflow, the better you'll do longterm, so check there early and often if you run into trouble.

We'll be using a few different terminals, and when you are meant to actively type something, I will try to copy the prompt: bash (my terminal) uses `$ `, Python uses `>>> ` and R uses `> `. Code with no prefix will be a `message printed back to you`.

Go to the appropriate section to set up your own environment for general purpose processing.

2.2.1 Ubuntu

We'll be using Ubuntu 12.04. On Ubuntu, you get the Advanced Packaging Tool, which we invoke with `$ apt-get`. We'll install a few other utilities off the bat:

Get git Start with `$ sudo apt-get install git`. Typing `$ git` should now output something reasonable. Bad news if you see `-bash: git: command not found`, and you'll have to do some research to fix this.

Get python Already built in – I'd just use theirs.

Get R We want 3.0.2, and for this we need to add their repository before we can `apt-get` R. Check out this page for good information.

2.2.2 OSX

Get Homebrew. Instead of `apt-get`, we'll use Homebrew. OS X ships with ruby, so you should be able to copy/paste the command at the Homebrew website with no problem.

Get git Now we can run `$ brew install git`. Typing `$ git` should now output something reasonable. Bad news if you see `-bash: git: command not found`, and you'll have to do some research to fix this.

Get python Python is already built into OS X (open your terminal and type `$ python` to start hacking away), but you want a different version of Python. Luckily, this is nice `$ brew install python`. This will install Python 2.7.6, which is what we'll be using. There are many discussions about Python 2 vs Python 3. We follow the advice of this one,

and use Python2 while writing code that will be Python3 compatible. The main differences are that division is no longer integer division (in python2, `>>> 3/4 = 0`, in python3 `>>> 3/4 = 0.75` but `>>> 3//4 = 0`), more objects are generators, and the `print` function is invoked like a function.Python 3.4 was just released, but `$ brew install python3` will still install Python 3.3.3. Use at your own (minimal) risk.

Get R As usual, `$ brew install R` installs a good version of R (I am on 3.0.2).

Chapter 3

Python

3.0.3 Python Interpreter Hello World

Open your terminal, type `$ python` and then `>>> print("Hello, World!")`.

That's a bit too easy. Let's instead run

```
$ python
Python 2.7.5 (v2.7.5:ab05e7dd2788, May 13 2013, 13:18:45)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = "Hello, World!"
>>> print(x)
Hello, World!
>>> print(x + x)
Hello, World!Hello, World!
>>> print(5 * x)
Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!
>>> print(5 * (x + "\n"))
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

Notice that you can assign strings to a variable, and manipulate strings using addition in multiplication (and it works in a sane fashion). The `\n` string is a newline. The other useful character to know for now is `\t`, which is a tab. To quit the console, `ctrl+d` or `>>> quit()`.

3.1 Cookbook

3.1.1 Serving files over a web server

This sounds intimidating, but you can literally go to any folder on your machine and run `python -m SimpleHTTPServer 8080`. Then if you open a browser and navigate to localhost:8080, you will see all the files in your original folder.

3.1.2 Serving a simple website with Flask, and Returning a list as JSON from a Flask app

A great setup for simple websites is Flask with bootstrap for css. While working on an app to plot JSON data in D3, I used the following

```
import random
from flask import Flask, Response
import json
app = Flask(__name__)

@app.route('/data')
def random_data():
  colors = ["red", "green", "blue"]
  data = [{
    "x": 10 * random.random(),
    "y": 10 * random.random(),
    "c": random.sample(colors, 1)[0],
    "size": random.randint(1,5)
    }
    for _ in range(10)]
  return Response(json.dumps(data), mimetype='application/json')

if __name__ == '__main__':
  app.run("127.0.0.1", port=5000, debug=True)
```

Now if I go to `127.0.0.1:5000/data`, I'll be served some json. `127.0.0.1` is another name for `localhost` (so `localhost:5000/data` will also do it). Setting `debug=True` lets me edit the script and it will auto-reload, as well as gives a helpful error page with an interactive python terminal if something goes wrong. One other small point about this: if the `data` object was a dictionary, I could have returned `jsonify(data)`, (where I would also need to `from flask import jsonify`), but for security reasons I need to return this one myself by converting it to json, and setting the proper mimetype. In this script, I create an `app` object, define one route, and serve json over that route.

3.1.3 Printing a number with commas

Check out PEP 3101 on new string formatting. `"${:,d}".format(10 ** 9)` outputs the string `'$1,000,000,000'`. The `{...}` indicates that there is something to be replaced, you can put a key (in this case `0`) before the colon to decide what will be replaced, the `d` says the object is an integer, and the `,` says to add commas every third digit.

Chapter 4

R

4.0.4 R Interpreter Hello World

Happily, we may again type `$ R`, and `> print("Hello, World!")`. We can't run exactly the same session as we did in Python, since R doesn't let us mix algebra and strings quite as nicely, but there are equivalents.

```
$ R

R version 3.0.2 (2013-09-25) -- "Frisbee Sailing"
Copyright (C) 2013 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin10.8.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> x <- "Hello, World!"
> print(x)
[1] "Hello, World!"
> print(paste(x,x))
[1] "Hello, World! Hello, World!"
> print(paste(rep(x, 5), collapse=""))
[1] "Hello, World!Hello, World!Hello, World!Hello, World!Hello, World!"
> cat(paste(rep(x, 5), collapse="\n"))
Hello, World!
Hello, World!
```

```
Hello, World!
Hello, World!
Hello, World!>
```

A few things to note about this:

1. We use the `paste` function to concatenate strings.

2. The `rep(x, 5)` created a vector with five copies of `x`. We could have done this by hand with `> c(x, x, x, x, x)`. More on R vectors later.

3. The `paste` function accepts a `collapse` argument, which tells it how to join a vector of strings.

4. In the last command, we used `cat` instead of `paste`. The `paste` function would print "\n", while the `cat` function interprets these as newlines.

5. The `cat` function doesn't automatically put a newline after it does its thing, which is why the interpreter starts typing after the last "Hello, World!"

4.1 Tools for using R

R has some fantastic support for getting up and working quickly and easily. A few ways to help get hacking:

Defining variables. R uses the symbol `<-` instead of `=`. Using `=` will still work (see this article if you're interested), but it is typically reserved for setting function arguments.

Using Packages You can import R code in a number of ways. I just use `> require(ggplot2)` to import, for example, ggplot2. You could also use `> library(ggplot2)`, with almost identical results. If I am importing a local file, use `> source('../relative/path/to/myfile.R')`. Note that with this, you can run code, assign values to variables, and define functions. A good workflow is to have an interactive console open, figure out how to make R do what you want to do, copy that over to a file, and `source` the file.

Installing Packages. A great strength of R is that it has some cutting edge packages available. These are uploaded to CRAN, the "Comprehensive R Archive Network", which is "a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R". The majority of packages you'll want are installed via `> install.packages("ggplot2")`. You'll be asked to choose a mirror, and typically the best will be the cloud mirror that RStudio maintains on an Amazon EC2 instance – number 0. It syncs with the central CRAN mirror in Austria once a day, and is typically beats any other mirror for speed (in Austin, the cloud mirror is faster than using the mirror in Dallas).

Occasionally, you'll want something really cutting edge that you find on github. There's a nice package from Hadley Wickham that makes this easy:

```
> require(devtools)
Loading required package: devtools
> install_github('ggplot2')
Installing github repo ggplot2/master from hadley
Downloading ggplot2.zip from https://github.com/h...
Installing package from /var/folders/k1/3qh0v1017...
arguments 'minimized' and 'invisible' are for Win...
Installing ggplot2
'/Library/Frameworks/R.framework/Resources/bin/R'...
  '/private/var/folders/k1/3qh0v1017ql8p4xm7hqzv_...
  --library='/Users/colinc/Library/R/3.0/library'...

* installing *source* package 'ggplot2' ...
** R
** data
*** moving datasets to lazyload DB
** inst
** tests
** preparing package for lazy loading
** help
*** installing help indices
** building package indices
** testing if installed package can be loaded
* DONE (ggplot2)
> require(ggplot2)
Loading required package: ggplot2
> sessionInfo()
R version 3.0.2 (2013-09-25)
Platform: x86_64-apple-darwin10.8.0 (64-bit)

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils
```

```
[5] datasets  methods   base

other attached packages:
[1] ggplot2_0.9.3.1.99 devtools_1.4.1

loaded via a namespace (and not attached):
 [1] colorspace_1.2-4   dichromat_2.0-0
 [3] digest_0.6.4       evaluate_0.5.1
 [5] grid_3.0.2         gtable_0.1.2
 [7] httr_0.2           labeling_0.2
 [9] MASS_7.3-29        memoise_0.1
[11] munsell_0.4.2      parallel_3.0.2
[13] plyr_1.8           proto_0.3-10
[15] RColorBrewer_1.0-5 RCurl_1.95-4.1
[17] reshape2_1.2.2     scales_0.2.3
[19] stringr_0.6.2      tools_3.0.2
[21] whisker_0.3-2
>
```

Note the use of `> sessionInfo()` at the end to view all the attached packages and their versions.

Getting help. It is notoriously difficult to google for help with R, given its name (though, really, who hasn't come up with unexpected results while searching for help with LaTeX, Python, or Ruby? I've even heard a story of a trip to manpages.com in search of bash help.) The built-in R help pages are accessed via `?`, and are actually quite helpful.

Note that `?` will only search packages which are currently loaded. Try the following session:

```
> ?geom_point
No documentation for  geom_point  in specified packages and
    ↪ libraries:
you could try    ??geom_point
> require(ggplot2)
Loading required package: ggplot2
> ?geom_point
```

The suggestion to use `??` is also a good one: when I type it in, I see

```
Help files with alias or concept or title
matching  geom_point  using regular
expression matching:


ggplot2::geom_point
               Points, as for a scatterplot
ggplot2::geom_pointrange
               An interval represented by a
               vertical line, with a point
               in the middle.
```

```
Type '?PKG::FOO' to inspect entries
'PKG::FOO', or 'TYPE?PKG::FOO' for entries
like 'PKG::FOO-TYPE'.




(END)
```

This command has searched through all *installed*, not necessarily loaded, packages. If you see a sweet function on a blog post and want to find which package it is in, you need to either go to google, RSeek, crantastic, or use the `sos` package:

```
> install.packages('sos')
Installing package into    /Users/colinc/Library/R/3.0/library
(as    lib    is unspecified)
--- Please select a CRAN mirror for use in this session ---
trying URL 'http://cran.rstudio.com/bin/macosx/contrib/3.0/sos_1
    ↪ .3-8.tgz'
Content type 'application/x-gzip' length 213172 bytes (208 Kb)
opened URL
==================================================
downloaded 208 Kb


The downloaded binary packages are in
  /var/folders/k1/3qh0v1017ql8p4xm7hqzv_3m84fy03/T//Rtmp3Dpn1e/
      ↪ downloaded_packages
> require(sos)
Loading required package: sos
Loading required package: brew

Attaching package:    sos

The following object is masked from   package:utils  :

    ?

> findFn('geom_point')
found 197 matches;  retrieving 10 pages
2 3 4 5 6 7 8 9 10

Downloaded 175 links in 50 packages.
>
```

This creates a local website with links to all the packages that you might find `geom_point` in.

Chapter 5

Bash

5.1 Cookbook

5.1.1 Count files in a folder.

If you have a lot of files in a folder, `$ ls -1` (that's the numeral 1) will list the files in the folder, one to a line, without a header. This is piped to a word count of lines.

```
$ ls -1 | wc -l
```

5.1.2 View head of gzipped file

Use `gzip`, with the `-c` flag to output to `stdout`, and `-d` to decompress (instead of compress). This can be piped to head.

```
$ gzip -cd | head
```

5.1.3 Dealing with corrupt gzips

Sometimes a process will be consuming a bunch of gzips and it will fail for some reason (this happened to me with an `unexpected end of file` error). To find the files that may fail, run `$ gzip -t`. To recover what part of the gzip you can recover, `$ gunzip < corrupted.gz > corrupted.partial`

Chapter 6

Scala

6.1 Recipes

6.1.1 Write a (json) string to file

We create a test `Map` named `x`, convert it to JSON, and then write it to a file called `test.json`. This will work for any `String`, but JSON is good to know how to deal with. We use the spray-json library to handle JSON objects, and fall back to a Java library to write to file. Here is a session in the REPL:

```
Welcome to Scala version 2.9.3 (Java HotSpot(TM) 64-Bit Server VM, Java
    ↪ 1.6.0_65).
Type in expressions to have them evaluated.
Type :help for more information.

scala> val x = Map("1" -> "a", "2" -> "b")
x: scala.collection.immutable.Map[java.lang.String,java.lang.String] = Map
    ↪ (1 -> a, 2 -> b)

scala> import spray.json._
import spray.json._

scala> import DefaultJsonProtocol._ // !!! IMPORTANT, else `convertTo` and
    ↪  `toJson` won't work correctly
import DefaultJsonProtocol._

scala> x.toJson.prettyPrint
res0: String =
{
  "1": "a",
  "2": "b"
}

scala> import java.io._
```

```
import java.io._

scala> val file = new File("test.json")
file: java.io.File = test.json

scala> val buff = new BufferedWriter
BufferedWriter

scala> val buff = new BufferedWriter(new FileWriter(file))
buff: java.io.BufferedWriter = java.io.BufferedWriter@6f84cdab

scala> buff.write(x.toJson.prettyPrint)

scala> buff.close()
```

Now to check everything went smoothly:

```
$ cat test.json
{
  "1": "a",
  "2": "b"
}
```

6.1.2 Print a list to screen

This shows how to either print and entire list with `foreach`, or just a few elements, with `take`.

```
scala> val x = List((1,2), (2,3), (3,4))
x: List[(Int, Int)] = List((1,2), (2,3), (3,4))

scala> x.foreach(println)
(1,2)
(2,3)
(3,4)

scala> x.take(2).foreach(println)
(1,2)
(2,3)
```

Chapter 7

Web Development

This covers dashboards and sharing data.

7.1 Dictionary

There are a lot of different technologies one might use.

7.1.1 SASS

"Syntactically Awesome Style Sheets", SASS is an extension of CSS3 that allows for things like variables and inheritance so that you can write sane CSS.

7.2 Cookbook

7.2.1 Installing NodeJS

Either use the installer from the website, or (on Ubuntu), run

```
$ sudo apt-get install python-software-properties
$ sudo add-apt-repository ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install nodejs
```

7.2.2 Using npm

NodeJS uses npm as its package manager (hence, NodePackageManager). You might like a package manager because different projects you work on depend on different versions of the same packages. If you want to install a package called `express` for a project, you can do so with `npm install express`. If you want all projects you run on your computer to be able to use express, then add a `-g` flag to install globally: `node install -g express`.

7.2.3 Starting an AngularJS App

First, install nodejs (subsection 7.2.1). We use Yeoman to scaffold an AngularJS app, and manage dependencies with Bower. We

1. Install these tools with npm,
2. install a Yeoman generator that actually scaffolds the angular app,
3. make a directory,
4. scaffold the app,
5. create a git repo

```
$ npm install -g yo grunt-cli bower
npm http GET https://registry.npmjs.org/yo
npm http GET https://registry.npmjs.org/grunt-cli
npm http GET https://registry.npmjs.org/bower

(this continues)

$ npm install -g generator-angular
npm http GET https://registry.npmjs.org/generator-angular

(more noise)

$ mkdir sample_project
$ cd sample_project
$ yo angular
(interactive stuff, noise)
$ git init
Initialized empty Git repository in /Users/colinc/sample_project/.git/
$ git add --all
$ git commit -am "Initialized an empty angular repo"

(noise)
```

Now you have a bunch of folders –

`Gruntfile.js` We use Grunt to manage packages (Grunt: the package manager for the web), and the Gruntfile is where we set up how Grunt does its job.

`app/` This is the folder where our app actually lives. Being javascript, it is a static app, so you could just drop in here and serve it with a Python SimpleHTTPServer (subsection 3.1.1).

Chapter 8

Algorithms

8.1 Sorting

Here is a summary of sorting algorithms

	in-place?	stable?	worst	average	best	remarks
Selection	×		$\frac{N^2}{2}$	$\frac{N^2}{2}$	$\frac{N^2}{2}$	N exchanges
Insertion	×	×	$\frac{N^2}{2}$	$\frac{N^2}{4}$	N	Use for small N or partially ordered
Shell	×		?	?	N	Tight code, subquadratic
Quick	×		$\frac{N^2}{2}$	$2N \ln N$	$N \lg N$	$N \log N$ probabilistic guarantee fastest in practice
3-way quick	×		$\frac{N^2}{2}$	$2N \ln N$	N	Improves quicksort in presence of duplicate keys
Merge		×	$N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee, stable
Heap	×		$2N \lg N$	$2N \lg N$	$N \lg N$	$N \log N$ guarantee, in-place

8.1.1 Dictionary

in-place This concerns memory usage: an in-place sort does not use a second array to do the sort.

stable This concerns composed sorts: if we had an array `[(a, 1), (a, 2), (b, 1)]` which is already sorted with respect to the first element in each tuple, an algorithm is stable if it guarantees that sorting with respect to the second key will return `[(a, 1), (b, 1), (a, 2)]`, instead of moving `(b, 1)` to the front.

8.1.2 Algorithms

Selection Sort

We iterate through the array, find the next smallest, and move that to the start of the list.

8.1.3 Insertion Sort

At each step along the array, we compare an entry with the neighbor to its left, and exchange the two if they are out of order.

8.1.4 Shell Sort

Like insertion sort, but use different gap sequences: perhaps compare each element with the neighbor 7 to its left first, then use a gap of 3, and finally 1. Wikipedia reports that a practical gap of $\lceil \frac{9^k - 4^k}{5 \cdot 4^{k-1}} \rceil$, which looks like $1, 4, 9, 20, 46, 103, \ldots$, is good. I have no intuition for why.