PYTHON FOR COMMUNICATION SCIENTISTS: FIRST STEPS TO AUTOMATED CONTENT ANALYSIS

by Damian Trilling

Version 0.1 – January 2014

0. Before we start: what & why?

"Big Data" is one of those often misused catchall terms. It sounds sexy, and people use it for all kind of data, but as a scientist, we want a clear definition, and it is hard to tell what Big Data actually entails. While I wrote this document in the context of our department's Big Data project, this document is not about *really* Big Data in the sense that it would require a whole server farm, but it is about data that is too big to handle manually – think of one million tweets for example. It is about data sets that are small enough to be handled by an ordinary desktop computer, but too big to be processed by ordinary programs. Excel does not have an unlimited number of rows, and SPSS and STATA start complaining (or simply stop working) once you have too many cases and variables.

This document introduces you to automated content analysis of such amounts of data: Tweets, blogposts, articles from RSS-feeds. We will use the programming language Python, which is very flexible and highly suitable for this end. It also *scales* very nicely – meaning that you can use it now for some smaller projects, but it can also be used on immense data sets.

This tutorial will guide you through the first steps towards automated content analysis with Python – I hope it can trigger your curiosity to continue your search for building the perfect analytical tool. There are enough further resources to help you.

Have fun!

Damian Trilling

d.c.trilling@uva.nl www.damiantrilling.net @damian0604

Special thanks to Mariella Bastian, Anne Brons and Peter Neijens for their feedback on a draft version of this tutorial.

Table of Contents

1. Preparing your equipment

Two principles have guided the choice of tools used in this tutorial: they should be *free* and – as far as possible – *platform independent*. For example, SPSS is not only incredible expensive if you aren't a student, it also does not run on Linux – and AMOS does not run on a Mac.

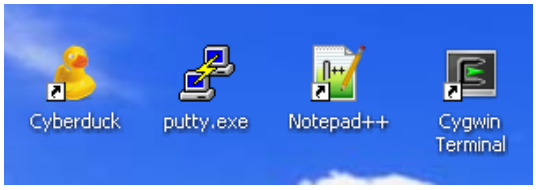

Figure 1. Important tools for Windows users.

In this tutorial, you are free to use whatever computer you want. Sometimes, you have to choose the right option for your specific environment. While writing this tutorial, I tried to avoid using platform-specific solutions. This is important in the context of big files, as you might – either now or in the future – be able to run your analyses on more powerful machines (on a server or in the cloud, for example).

In the next section, you'll learn about the tools you should install.

1.1 A good text editor

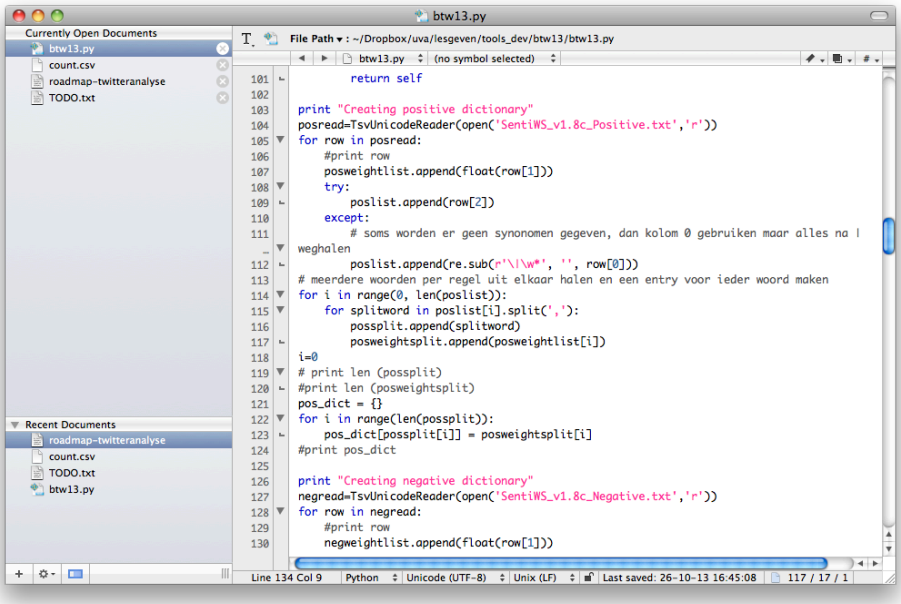

Figure 2. A good text editor (in this case, TextWrangler) is essential. It automatically recognizes if you write Python code and then uses different colors to mark variables, strings, functions, comments and so on.

Basically, you could use anything to write Python code or to inspect plain text data files – even the standard Windows Notepad will do the job. However, a good editor makes life much easier. It allows you to choose different encodings (important when dealing with characters like ä, ø or even €), highlights your syntax, shows line numbers, allows for sophisticated search- and replace actions and much more. For Windows, Notepad++ is a good choice (download it from http://notepad-plus-plus.org/), Mac users might like Textwrangler (download it from http://www.barebones.com/products/textwrangler/). Emacs is a good allrounder on each system (Linux, Mac, Win) as well (download it from http://www.gnu.org/software/emacs/).

1.2 Python

Unlike a program like SPSS (where you start a program and then work within it), Python does not come with a graphical user interface (actually, there is one, but you still have to write the code yourself and it's often not the smartest thing to use it). You can write your code in whatever program you want (see 1.1 A good text editor), and Python runs it for you.

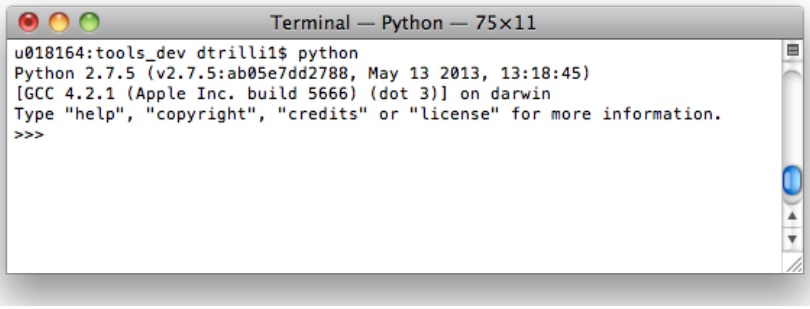

Figure 3. Starting Python: You just enter "python" and press Enter.

It is crucial to have the right version: Do **not** use Python 3.x, it is a different language than 2.x! Python 2 is still much more commonly used than Python 3, this is why we use that version. At this moment, Python 2.7.6 is the most recent version of Python 2.

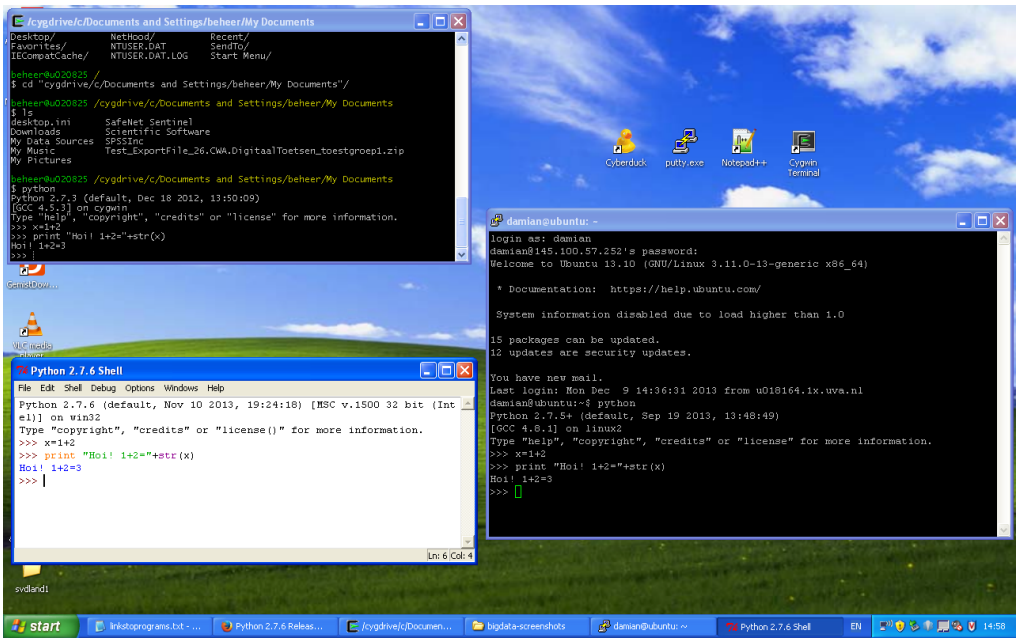

Figure 4. Three ways of running Python on Windows: In Cygwin (top left), Python downloaded from python.org (bottom left), or via SSH on our server.

MacOS already comes with Python 2.6 installed. Open the terminal (you find it in the Utilities folder within the Applications folder), type `python` and you'll get something like Figure 3. Exit by typing `quit()` to leave Python so that you can use the Terminal for other tasks again. Have a quick look at the version number: 2.6 should be fine, although some examples require Python 2.7 (which is why I, as you can see in the figure, updated mine).

Windows users have to download Python from http://www.python.org/ (where Mac users also can get their update, if they want to). If you have no idea which of the many options offered is right for you, than most likely something like "Python 2.7.6 Windows Installer (Windows binary -- does not include source)"

(http://www.python.org/ftp/python/2.7.6/python-2.7.6.msi) will be fine for you.

Alternatives are using Cygwin (see 1.4 Common UNIX-tools) or using Python on our server (see 1.3 Tools to connect to our server).

1.3 Tools to connect to our server

We have a Linux-server on which you can log on (request access by mailing the instructor of your course). This is done via a protocol called SSH. It stands for "Secure SHell" and basically means that you log on to another computer and work on it as if it was your own.

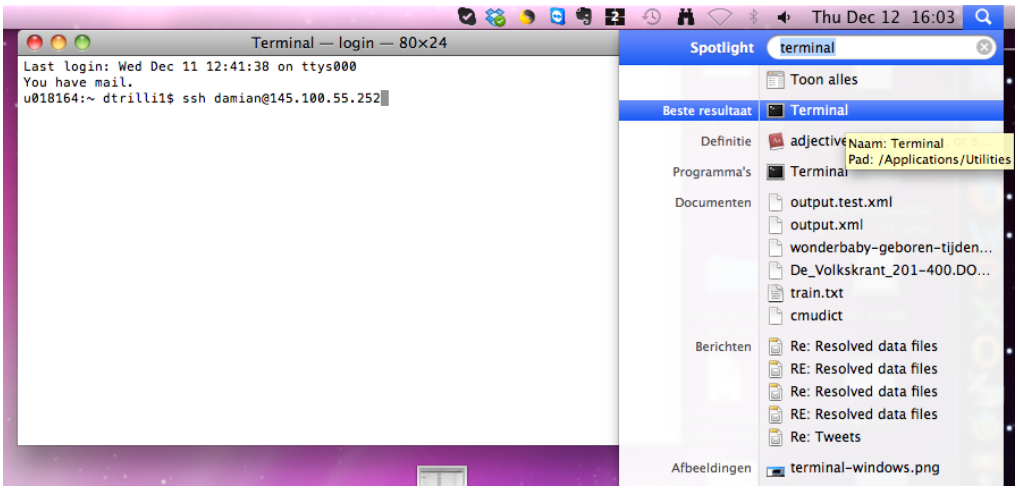

Figure 5. Accessing a Server via SSH on a Mac

Mac users can simply open the Terminal (just search for it with Spotlight) and enter the command

```
ssh damian@speeltuin.followthenews-uva.cloudlet.sara.nl
```

(if your username is damian and you want to connect to the server speeltuin.followthenews-uva.cloudlet.sara.nl).

On Linux (or in Cygwin, see the next page), it works essentially the same way.

Windows users have to download a small tool called PuTTY (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html).

Bonusbox: Transferring files

To transfer files to and from the server, you can use such command line tools as well (and sometimes, that is the smartest solution), but to make life easier, you might additionally want to install a tool like WinSCP (http://winscp.net) or Cyberduck (http://cyberduck.io/). It also allows to transfer files using the FTP standard, something that might come in handy as well. You receive account information for this when you request access to our server.

1.4 Common UNIX-tools

If you just want to have a quick start, skip this section. Many of you are probably Windows users, but Linux (and basically also MacOS) provide you with a huge amount of build-in commands that might save you a lot of work. Later in this tutorial, I will sometimes refer to tools that are available on these operating systems, so if you think that they are useful to you (and I'm sure they will be), go back here.

UNIX-like operating systems (Linux, but also MacOS) come with a huge amount of useful command-line tools that are useful for retrieving and processing data.

So what to do if you are a Windows user?

The first option is to log on to our Linux-Server – on which all tools are installed already – and work on the server instead of on your own computer. This only requires a SSH-client (see 1.3 Tools to connect to our server).

But if you want to make use of all the useful small Linux-tools on your own Windows-machine, it might be useful to install Cygwin (http://cygwin.com), an environment that emulates a Linux environment on Windows. You can even choose a great amount of packages, so that for example you have Python already included in your Cygwin-environment, so that you can run it there as well.

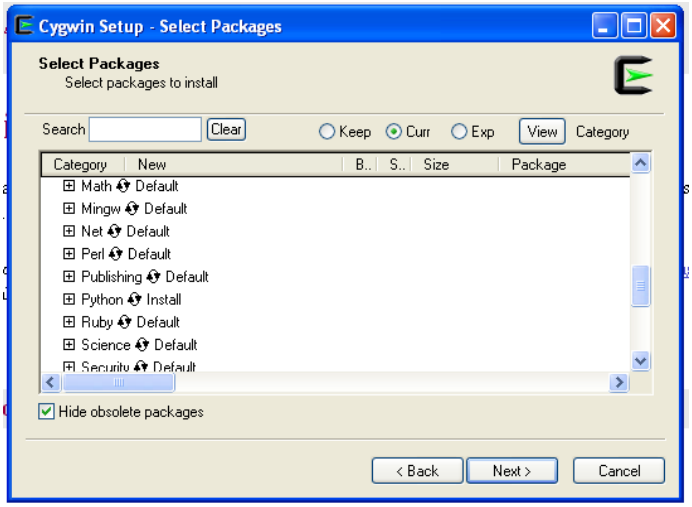

Figure 6. Selecting packages (Python!) to install with Cygwin.

2. Getting to know the Command Line

This chapter provides you with some basic knowledge about how to use the command line of your operating system. Chances are very high that you will need this later, so before diving into Python (Chapter 3), I recommend doing this first.

2.1 Navigating the directory tree

Let's get started with a generational divide. It greatly helps if you are of the age of the author (30) or older, as you probably remember the thing in Figure 7. If you have never seen something like this before (its called a "command line" or a "prompt"), catch up watching a video:

Figure 7. Who remembers this?

- Windows users: http://www.youtube.com/watch?v=MNwErTxfkUA
- Mac users: http://www.youtube.com/watch?v=yIY3iPDVUBg
- Linux users: (OK, you probably know this anyway…)

Let's start and create a directory (a folder) for all the exercises we are going to do in this tutorial.

Windows	MacOS (and Linux)
1. Open the Command line prompt	
Click start button, enter `cmd` in the search field and press enter.	Click the spotlight-icon in the upper right corner, enter `Terminal` and press enter
2. Check where you are.	
The prompt should show something like c:\users\brigitte. This is where you are.	Enter "pwd" and press enter. You should get a response like /Users/dami This is where you are.
3. Create a sub-directory	
Enter `md pythontutorial`	Enter `mkdir pythontutorial`
4. Go there!	
Enter	Enter

`cd pythontutorial`	`cd pythontutorial`
Wanna be sure if you're really there? See step 2.	Wanna be sure if you're really there? See step 2.
Enter	Enter
`cd ..`	`cd ..`
to go back to the parent folder. Note that there is a space between cd and .., but not between the two dots.	to go back to the parent folder. Note that there is a space between cd and .., but not between the two dots.
Now you'll learn a trick: enter	Now you'll learn a trick: enter
cd pythontu (but instead of completing it, press the TAB key to let the computer complete it). Press Enter.	cd pythontu (but instead of completing it, press the TAB key to let the computer complete it). Press Enter.
5. Check what's in the directory	
Enter	Enter
`dir`	`ls`
(OK, of course there's nothing in there yet)	(OK, of course there's nothing in there yet)
6. Let's create a file, check if it is there and display it	

`echo Hello world > test.txt`
`dir`
`type test.txt`
	`echo Hello world > test.txt`
	`ls`
	`cat test.txt`

Enough playing around. Let's get started.	

3. Your first Python script

3.1 Running Python in interactive mode

Start the python interpreter by typing “python” in the Terminal/Command line. Or connect to the server with SSH and type “python” once you have logged on.

> **Windows users: use the correct path!**
>
> Windows user have to type either the complete name instead (c:\python27\python if you installed Python in c:\python27, which is the default) or tell the command line once where Python is located. This is done by the following command:
>
> ```
> set path=%path%;C:\python27
> ```
>
> Remember this for future exercises!

Python starts up and welcomes you with

```
>>>
```

meaning that it waits for your input.

Enter

```
print “Hello world”
```

Congratulations! You just issued your first Python command. You might have noticed that “print” has nothing to do with your printer, it just “prints” something to your screen. OK, that isn’t very spectacular, so let’s go further. What about:

```
greeting=”Hello world”
print greeting
```

You just created a variable called greeting and displayed it on the screen. It contains text, therefore it is called a *string variable*, indicated by the quotation marks. Of course, you can also create numerical variables:

```
x=1+2
print x
```

It is getting more useful, isn’t it? You can not only add numbers, but also “add” two strings:

```
firstname=”Damian”
lastname=”Trilling”
fullname=firstname+lastname
print fullname
```

As you see, it gets better if we add a small string containing a space character in the middle:

```
fullname = firstname + ” “ + lastname
```

Or do something like this:

```
fullname=lastname+”, “+firstname
```

What you cannot do is “adding” a number to a string (this would not be very logical, would it?). You cannot write

```
print “One plus two is”+x
```

But there is a trick: The function str() changes a number to a string: It basically changes the value 3 to the character “3”. So we can write:

```
print “One plus two is ” + str(x)
```

or even:

```
print “One plus two is ” + str(1+2)
```

str() is an example of a *function*. Between brackets, you give an argument (in this case, a numerical value) and it returns some result (in this case, a string).

Python has a lot of nice build-in functions. One example:

```
print len(greeting)
```

gives you the number of characters of the string variable greeting that you created before. Of course, you could also go the direct way:

```
print len(“Hello world”)
```

Have you noticed the quotation marks? This is very crucial to help Python distinguish between variable names and the strings themselves.

You will learn later how you can use much more powerful functions like this to, for example, determine the sentiment or the grammatical features of a string. The nice thing: They are as easy to use as this one!

Of course you can combine everything you just learned:

```
y=x*len(greeting)

print str(x) + ” multiplied with “ + str(len(greeting)) + ”
equals “ + str(y)
```

What we are doing right now is called running Python in *interactive mode*: You enter a command, Python reacts. You might have guessed already that this is not very practical for writing a real program (it is very good for quickly trying something out, though). So let’s exit python with

```
quit()
```

and move on.

3.2 Running python programs from the command line

Let us do the same small exercise that we just did in a non-interactive mode (which is what you will do in 95% of all the time).

Open the text editor of your choice (see 1.1 A good text editor), enter the two lines

```
greeting=”Hello world”
print greeting
```

and save the file as helloworld.py in the folder you created in *chapter* 2. . Don’t call it helloworld.py.txt or something like that (Windows often hides file extensions or adds them without asking. This will become very annoying, as you need to know the *exact* file name. So, change your Windows-settings to avoid this. Google offers plenty of help (“display file extensions windows”)).

Go to the terminal, change the directory to the folder you created (2.) and enter

```
python helloworld.py
```

That’s all!

Windows users: Remember to use the correct path (see page 11).

Bonusbox: The #! statement on Linux and MacOS

(Windows is unfortunately not capable of interpreting the #!-statement, sorry guys!)

You could always run your programs with “python programname.py” as you just learned. However, this is no nice programming style. You want users to run your program by *only* entering the program name – in fact, they do not even need to know that it is written in Python and you do not want the user to care.

If you do not want to type

```
python helloworld.py
```

every time you run your script, but simply

```
./ helloworld.py
```

(if you already are in the right directory)

or

```
/Users/dami/pythontutorial/helloworld.py
```

(wherever you are – isn’t that cool?), then you just have to tell your computer where Python is located in the first line of the script itself. This is commonly done and considered good programming style, as it ensures that the right version of python is used if you have several versions installed – which actually is pretty often the case. Our version of the script would therefore rather be

```
#!/usr/bin/python
greeting=”Hello world”
print greeting
```

(/usr/bin is most likely the directory in which python is located).

I have for example one script starting with

```
#!/Library/Frameworks/Python.framework/Versions/2.7/bin/python2
.7
```

and another one starting with

```
#!/usr/bin/python2.6
```

because I want them to use a specific version of Python, but do not want to have to explain this to the user (and probably would not even remember myself).

One step is still necessary once you created helloworld.py to run it this way. Unix-like operating systems (MacOS, Linux) have quite rigid restrictions on whether one is allowed to (r)ead, (w)rite or e(x)ecute a file. By default, you are forbidden to execute (program) files directly from the command line. To allow yourself to run your script, enter

```
chmod u+x helloworld.py
```

Make sure you `cd` 'ed to the right directory, or use

```
chmod u+x /Users/dami/pythonturial/helloworld.py
```

instead.

3.3 Running python programs with the GUI

For Windows users, another comfortable way of running Python is using the graphical user interface (GUI). In the right window, you can write your program, open and save it, and see the output in the left window (Figure 8). If you like it, use it.

I personally prefer running python programs on the command line (see 3.2 Running python programs from the command line) for two reasons: You often will use the command line for other purposes during your programming session as well, for example to inspect the output of your program. And, maybe even more important, the build-in text editor is much worse than the editors I present in 1.1 A good text editor.

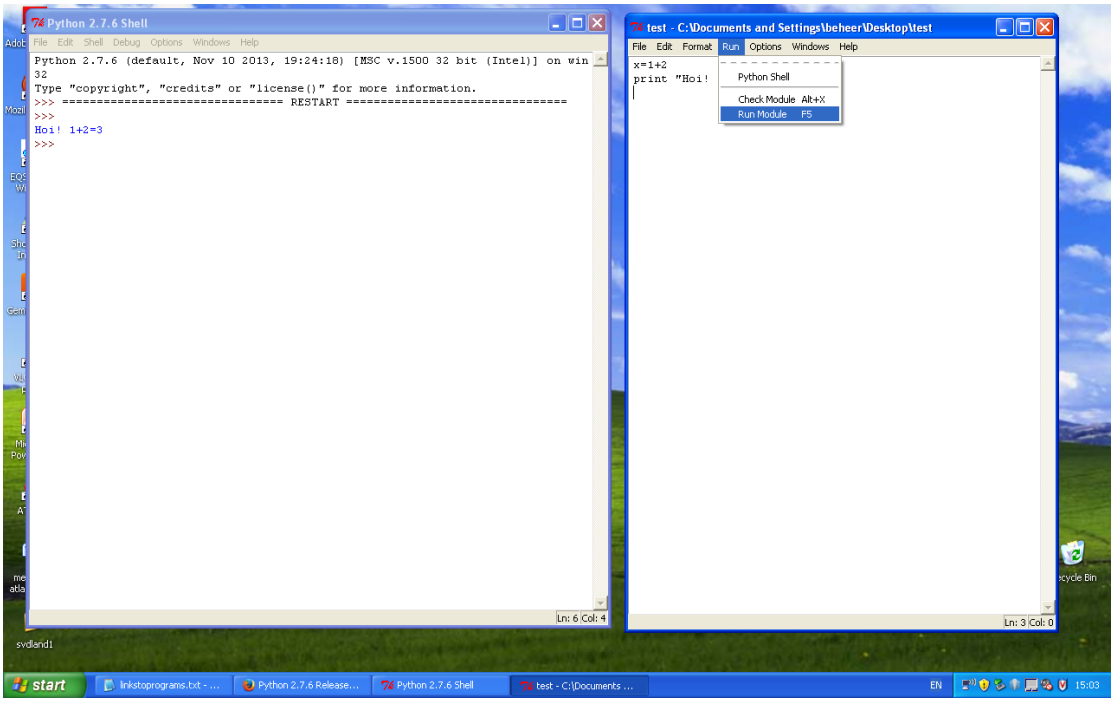

Figure 8. The graphic user interface (GUI) to Python on Windows. You can write the program in the right window, click "run" and see the output in the left window.

And, for the sake of completeness: The project IPython tries to make working with Python more interactive and contains a more advanced interface. It can be downloaded from http://ipython.org/ and works on Windows, MacOS and Linux. This is especially useful if you plan to make many visualizations, but is slightly different from “pure” Python and therefore falls beyond the scope of this manual.

Pérez, F., & Granger, B.E. (2007). IPython: A system for interactive scientific computing, *Computing in Science and Engineering, 9*(3), 21-29. doi:10.1109/MCSE.2007.53

4. Reading data, counting things

You now know how to run a Python program and you should have gotten a very basic understanding of how a program file looks like. You do not need to become a programmer, so we'll skip the technical details and start right away with a practical example. All you need to do is to download the code, save it in the folder you created before and modify it to your needs with your favorite text editor. To help you, I inserted a lot of lines starting with the #-sign. This means that these lines are comments to explain the code to you and ignored by Python (you know this from the *-signs in SPSS or STATA).

4.1 Input data: A CSV-file

If you collected Tweets with yourTwapperkeeper on our server datacollection.followthenews-uva.cloudlet.sara.nl, you received a so-called csv-file with all the tweets. It is like an Excel-Table (and you can easily open and save these files with Excel), but simpler: It's just plain text, where a comma (or a semicolon) indicates that a new column begins. Save the tweets you want to analyze in your pythontutorial-folder and call it mytweets.csv.

Open the file in your favorite text editor or learn how to use the command line for this from the Bonuxbox.

Bonusbox: Inspecting files on the command line

You already learned a number of commands you can use on the command line. There are also some commands that you can use to display files. The general idea is that you type the command followed by the file name that you want to show. Again, this only works for MacOs and Linux (or Cygwin).

You have three tools at your disposal: `head` (which gives you the first lines of a file), `tail` (last lines), and `cat` (all lines). If you want to learn more about a command like tail, just type `man tail`. By default, you get 10 lines, but you can specify a different number, thus, you can produce the output below this box with `head -3 mytweets.csv`.

But it gets even cooler: You can also just display a specific column (e.g., only the tweets): `cut -f1 -d, mytweets.csv` . See `man cut` for more details. And you can combine these kind of commands through a technique called "pipe", indicated by the | -sign. `cut -f1 -d, mytweets.csv | tail -10` for example sends the output of the cut-command to the tail-command. If you want to send output to a file instead of the screen, this is done with the > -sign (remember from page 10?). Try `cut -f1 -d, mytweets.csv > mytweetsonlyfirstcolumn.csv` !

mytweets.csv looks basically like this:

```
text,to_user_id,from_user,id,from_user_id,iso_language_code,so
urce,profile_image_url,geo_type,geo_coordinates_0,geo_coordina
tes_1,created_at,time
```

```
:-) #Lectrr #wereldleiders #uitspraken #Wikileaks #klimaattop
http://t.co/Udjpk48EIB,,henklbr,407085917011079169,118374840,n
l,web,http://pbs.twimg.com/profile_images/378800000673845195/b
47785b1595e6a1c63b93e463f3d0ccc_normal.jpeg,,0,0,Sun Dec 01
09:57:00 +0000 2013,1385891820

Wat zijn de resulaten vd #klimaattop in #Warschau waard?
@EP_Environment ontmoet voorzitter klimaattop @MarcinKorolec
http://t.co/4Lmiaopf60,,Europarl_NL,406058792573730816,3762391
8,en,<a href="http://www.hootsuite.com"
rel="nofollow">HootSuite</a>,http://pbs.twimg.com/profile_imag
es/2943831271/b6631b23a86502fae808ca3efde23d0d_normal.png,,0,0
,Thu Nov 28 13:55:35 +0000 2013,1385646935
```

The first column contains the tweet, the second the user that the tweet is directed at (which obviously can be empty), the third the sender and so on.

We will now write a Python script that reads the columns we are interested in, look for a search string, and write an output file with an additional column that contains the outcome of our search.

4.2 The script

Before you start, make sure to save unicsv.py (download it from https://github.com/uvacw/py-examples) in your pythontutorial-folder. It provides an easy method for reading CSV-files (the standard method does not support Unicode, meaning that you would run into problems if your tweets contain something like ä ø or €). Also download tutorial42.py, which is the script below, so you don't have to type it over.

```
#!/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7

# The line above is a typical starting line of a Python script.
# The #!-statement tells the computer which version of Python to use
# (if you do not know the path, you can leave it away,
# but it is good programming style to use it).

# Our script uses the following two modules from the file unicsv.py that you saved in
# the folder of this script. They are necessary to read and write our CSV-files.
from unicsv import CsvUnicodeReader
from unicsv import CsvUnicodeWriter

# We also want to import a module to search for Regular Expressions
# (an advanced form of a search string)
import re

# Let us define two variables (inputfilename and outputfilename) that contain
# the names of the files we want to user later in the script.
# Obviously, the input file should exist and contain the tweets (see the tutorial),
# the output file does not have to exist yet. Just choose a nice name.
inputfilename="mytweets.csv"
outputfilename="myoutput.csv"

# Now we create some empty lists that we will use later on. A list can contain several
# variables and is denoted by square brackets.
user_list=[]
tweet_list=[]
search_list=[]
iso_language_code_list=[]

# What do we want to look for? Between the two quotation marks we put a so called
# regular expression. Google gives plenty of help if you have difficulties with the syntax
# of such an regexp.
searchstring1 = re.compile(r'[Pp]olen|[Pp]ool|[Ww]arschau|[Ww]arszawa')

# Let the program start! We tell the user what is going on...
print "Opening "+inputfilename
# ... and call the function to read the input file.
reader=CsvUnicodeReader(open(inputfilename,"r"))

# Now we read the file line by line. Note the indention:
# the indented block is repeated for each row (thus, each tweet):
for row in reader:
        tweet_list.append(row[0])
        # we just append something to our empty list of tweets, namely the 1st
        # column of our row.
        # Note that we start counting with 0.
        # The ISO language code is in the sixth column, this is why we append segment 5
        # of our row to the iso_language_code list.
        iso_language_code_list.append(row[5])
        user_list.append(row[2])
        # we don't care about the other columns right now, but you now know how to add them:
        # make an empty list before, and then append appropriate row[column]
        # Let us count how often our searchstring is used in the tweet:
        matches1 = searchstring1.findall(row[0])
        matchcount1=0
        for word in matches1:
                matchcount1=matchcount1+1
        search_list.append(matchcount1)

# OK, we repated the procedure above for each row. Time to put all the data in one
```

```
# container and save it:
print "Constructing data matrix"
outputdata=zip(tweet_list,user_list,iso_language_code_list,search_list)
headers=zip(["tweet"],["user"],["iso"],["how often is Poland mentioned?"])
print "Write data matrix to ",outputfilename
writer=CsvUnicodeWriter(open(outputfilename,"wb"))
writer.writerows(headers)
writer.writerows(outputdata)
```

There is much to improve (for example, as you might have noticed, we analyzed the column headers from the input file rather than skip them), but for the sake of clarity and brevity, I decided to leave this for later.

4.3 Counting most frequently used terms

-- This example only works with Python 2.7 --

Imagine you want to rank the users from the example before and get a CSV file that makes a list with the users that wrote the most tweets (or with the frequencies of the words used, ...).
We will expand the script and have it produce a second CSV file, called whotweetsthemost.csv. It should contain the number of tweets produced by each user, beginning with the most active user. It will look like this:

```
12,dewereldmorgen
5,Trendingnewsnl
4,RobbertCasier
4,ChrstphDbrbndr
4,IPSVlaanderen
3,ceesnu
3,Klompsma
3,SimonSuys
2,ActionAid_NL
```

Take the script you wrote in the section before. First of all, we have to import a function called Counter at the beginning of the script. As the name says, it provides a comfortable way to count things, and this is why we need Python 2.7, as the earlier versions do not include this function.

```
from collections import Counter
```

Then, we define a third filename:

```
usercountfilename=”whotweetsthemost.csv”
```

At the very end, of the script, we insert:

```
counter=Counter(user_list)

with open(countuserfilename, "wb") as f:

      for k,v in counter.most_common():

            f.write( "{},{}\n".format(v,k.encode("utf-8")) )
```

What did we just do? Counter produces something called a dictionary with a *k*ey (the username) and a *v*alue (the number of tweets). For each **k**ey, the function writes the **v**alue, followed by a comma and the key into the file. The “.encode("utf-8")” makes sure that we use Unicode-encoding to avoid running into problems with special characters.

5. Sentiment analysis

There is much to say about sentiment analysis, the analysis of the tone of a text – you can have endless discussions about the pro's and con's of different approaches and the method itself. Nevertheless, it's frequently used and it does not hurt to get some training, so we'll build a small rudimentary analysis tool, based on the so-called bag of words approach: For now, we don't care about any linguistic features, but simply look at the words used.

5.1 Method 1: Comparing tweets with lists of positive and negative words

There is a great tutorial called "An introduction to text analysis with Python" by Neal Caren (http://nealcaren.web.unc.edu/an-introduction-to-text-analysis-with-python-part-1/). It teaches you – even without knowing Python yet – how to use Python to do a sentiment analysis of tweets by splitting up tweets in words, compare them with lists of negative and positive words, and calculate a positivity score. I couldn't explain it better, so I won't try but urge you to read that tutorial instead.

Such lists with positive and negative words exist and can be downloaded, but of course you could also make your own lists of different things you want to count (like names, synonyms,...) to answer more complicated questions like "Do tweets by left- and right-wing politicians differ in terms of negativity *when they use words referring their opponent*?" – which actually is what one of my Bachelor students did in her thesis.

5.2 Method 2: Using a module for sentiment analysis

One of the main advantages of using Python is that you can make use of many so-called modules. Once you installed them, they provide you with nice new functions (STATA and R users now that idea.)

One of this is the Pattern package (De Smedt, T. & Daelemans, W. (2012). Pattern for Python. *Journal of Machine Learning Research*, 13, 2031–2035). Download it from http://www.clips.ua.ac.be/pattern. The downloaded zip-file contains a file called README.txt that explains how to install it. Once installed, a sentiment analysis only takes two lines in Python:

```
from pattern.nl import sentiment
print sentiment('Een onwijs spannend goed boek!')
```

The first line tells python to make the function sentiment from the module pattern.nl (there are also pattern.en, .es, .de, .fr, .it) available. Then, you can use sentiment like any other function (remember the functions str() and len() that you already learned?).

This example from their website returns a *polarity score* between -1.0 and +1.0, and a *subjectivity score* between 0.0 and 1.0. In this case, 0.69 and 0.90.

Just play a bit around with it.

Bonusbox: Useful packages for language processing

If you browse a bit around the CLIPS-website, you see that pattern has a lot more great things to offer.

Another package that does similar things like pattern is the Natural Language Processing Toolkit (Bird, S., Klein, E., Looper, E. (2009). *Natural language processing with Python: Analyzing text with the Natural Language Toolkit*. Sebastopol, CA: O'Reilly). You can download the package (and the whole book!) from http://nltk.org.

Now it's up to you: Combine your knowledge from all the previous chapters and build a sentiment analysis tool that reads files, analyses them and produces an output file!

6. Looking forward

You now should have gotten a basic understanding of how Python works. We are collecting useful scripts that you can modify for your needs at https://github.com/uvacw/py-examples . You might also want to check out my website http://www.damiantrilling.net.

This tutorial is work in progress, so I would very much like to hear from you and receive any kind of feedback.

And of course, this tutorial only provides a small glimpse into the possibilities of Python for content analysis. I hope it triggered your interest – because now, you should have learned enough to look for bits and pieces and assemble them yourself to build your personal analysis tool.

Good luck!

Damian