Week 9 Laboratory Activity

Activity: Wrangling Text Data

Exploring Tweets on the command line

In this activity we will explore some Twitter data. Note, there are excellent Python libraries for obtaining and processing Tweets. In this exercise, however, we are given a file of Tweets that somebody else has downloaded in a particular format, and we have to work out how to process it.

Conventions Used

The majority of you will be using Windows 10 machines, while there are some who will be using MacOS (different versions) and maybe even fewer with Linux machines. As such, this activity sheet will use the common term CLI (command line interface) to refer to:

- **[Windows user]** Cygwin64 Terminal (for Windows, accessible from your Start Menu after you install it. You will need to install it; it is not part of Windows)
- **[Windows User]** Alternative: Bash terminal (Available via Start menu by typing 'Bash') if you setup according to lecture's content.
- **[MacOS]** Terminal (for MacOS, accessible from the Utilities folder in Applications)
- **[Linux users]** Terminal (for Linux users)

Do note that the folder (or usually referred to as "directory" in BASH scripting) locations on the different OS will be different and by default, this activity sheet will be based on the Cygwin64 environment (Windows environment).

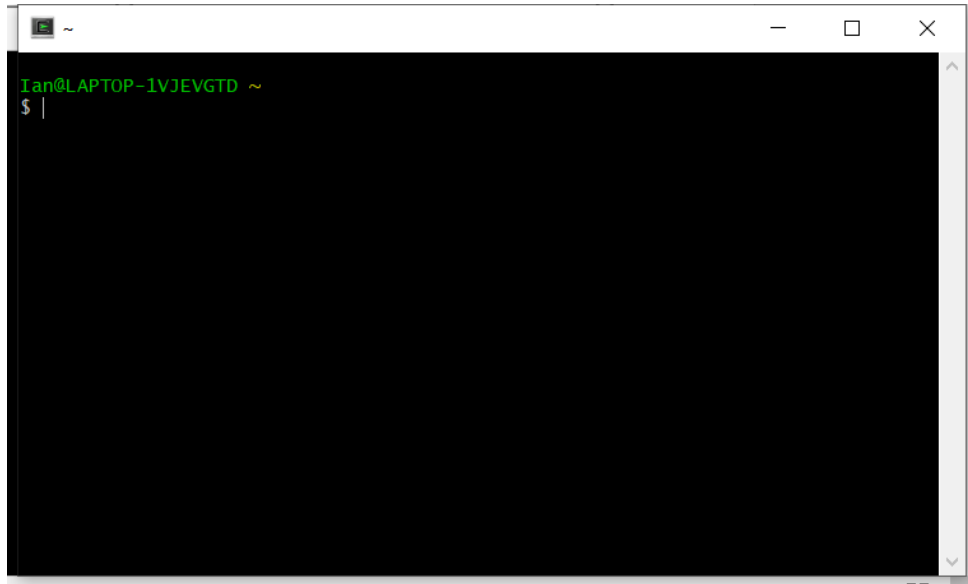### Getting and Checking the file

Download the `"msgraw_sample.txt.gz (gzipped Twitter data, 46Mb)"` file from Moodle.

Start your command line terminal (it should be something similar as below). We will refer to this as the BASH terminal or the CLI terminal.

Note: MacOS Big Sur (or later) Users

If you are on MacOS (Big Sur or later), you will need to run an extra command as your default shell is not BASH any longer. Pre-Big Sur's default is BASH (so, there isn't a need to do anything more). In your MacOS terminal, type
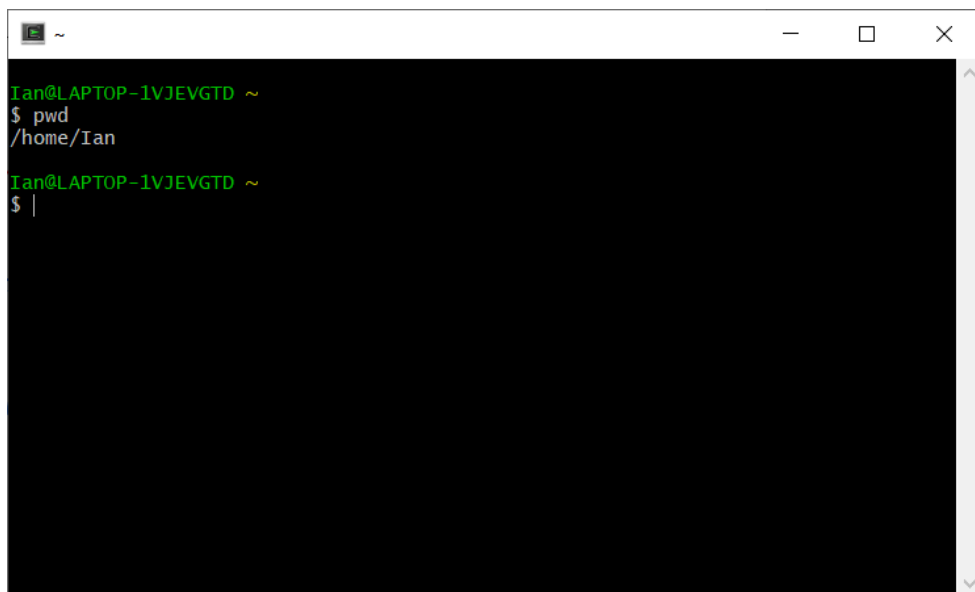
```
chsh -s /bin/bash
```

Important: If you ever get into trouble on the command line (for example you get trapped in some program and don't know how to quit, just type `<control>-c` to kill the program).

The first this you may want to check is the current folder you are in:

`pwd`

In my case, it shows `/home/Ian`. In all cases, the first thing that you need to do for this tutorial is to move / copy the msgraw_sample.txt.gz file to the appropriate location Assuming that the file `msgraw_sample.txt.gz` is downloaded in your default download location, you can move or copy the file to your current folder (called the "working directory").

`mv /cygdrive/c/Users/Ian/Downloads/msgraw_sample.txt.gz .`

You can substitute the "`mv`" with "`cp`" if you want to make a copy.

Check the size of the file using the `-lh` option with the command `ls`:

```
ls -lh msgraw_sample.txt.gz
```

Uncompress the file using the "gunzip" command:

```
gunzip msgraw_sample.txt.gz
```

Practise 1: Now how big is it? Use the command below.

ls -lh msgraw_sample.txt

Practise 2: So what is the compression rate as a percentage (uncompressed/compressed*100)% ?

Look at the first few lines:

```
head -5 msgraw_sample.txt
```

Looks like a bit of a mess? The file certainly has long lines! You may want to widen the shell window so that you can see more of each line.

The first line looks to be a header. Let's check:

```
head -1 msgraw_sample.txt
```

Yes, it is a header for a CSV file, probably. The entry "`7RTcount`" means the (7+1)-th column has the label "`RTcount`", which is probably the retweet count (i.e. the number of times other Twitter users have reposted the message).

Some of the other columns seem to contain metadata like "`10Geo#`" (geocoding data), "`2CreatedM`" (perhaps tweet creation time?), "`16Name`" (the author's user name), "`22Followers%`".

Since the file is probably a CSV, a critical thing we need to know is what is the field delimiter. There seems to be uneven spacing, which suggests it may be a tab character.   To check this we can send the first line to "`less`":

```
head -1 msgraw_sample.txt | less
```

And show all the tabs in the line by entering "`/<tab>`" where `<tab>` denotes the tab key on the keyboard (to check, once you have done it, it should show a `^I`). You should see all the tabs in the file light up (probably appear as white areas). And yes, we see that tab is indeed the delimiter being used to separate the column values.

When you're done, type "`q`" to quit `less`.

How big is the file in lines? Use "`wc`" which does a word count, and the "`-l`" flag (note that the "`l`" is not the number 1 but the lower case character L) to just report the lines:

```
wc -l msgraw_sample.txt
```

Practise 3: How many are there?

Inspecting the Fields / Columns

Now let's inspect some individual fields. The unix "`cut`" command delimits lines with tabs, so it is perfect to use on this file. We tell cut which columns to output using the `-f` flag, so if we want to see column 3 for instance, we would write "`-f 3`". Let's do that, but since the file is very long, we won't print out all the lines to the screen, but rather **pipe** the results to "`head`" and just report the first 20 lines:

```
cut -f 3 msgraw_sample.txt | head -20
```

Note that in the Unix shell there are many ways to perform the same operation. In order to extract the third column from the file, we could have also used the "`awk`" utility, we would just need to tell `awk` that the delimiter used is the tab character denoted '`\t`':

```
awk -F '\t' '{print $3}' msgraw_sample.txt | head -20
```

Or we could even use a very short Perl script (not installed by default on your Cygwin, and there is not need to do so) to do the same thing:

```
perl -n -e '@a=split(/\t/,$_); print "$a[2]\n";' msgraw_sample.txt | head -20
```

So we can see that a simple use of "`cut`" and more complex uses of "`awk`" and "`perl`" could do the same thing. Awk is a simple extraction language whereas Perl is a full scripting language rather like Python. Perl is older than Python and in some sense, Perl was the motivation for developing Python. Perl itself is really a clean-up of shell scripting, trying to incorporate capabilities like awk.

Which should you bother to learn? Well, Python does most of these tasks and is generally considered better than Perl, except on some string processing where Perl can be faster and easier to write. But nevertheless, "`cut`", "`awk`" and "`sed`" have some tremendous short commands that let you do a lot, so it is worthwhile learning their basics along with basic shell scripting (called "`bash`"). Perl is probably best ignored unless you want to focus on a lot of string handling.

Back to the file, let's have a look at the first column which contains the text of each tweet. This time we'll pipe to "`less`" so we can look forwards and backwards in the file (by using the `<up>` and `<down>` keys).

```
cut -f 1 msgraw_sample.txt | less
```

Looks interesting! There are a few different languages in there (Spanish, Dutch,...). Search for the string "Melbourne" to see what people are saying on Twitter. To search, type "`/Melbourne`". To repeat the search (for the next instance of the string), just type "`/`". We could also just print out all of the tweets containing the term "`Melbourne`" by piping to "`grep`":

```
cut -f 1 msgraw_sample.txt | grep "Melbourne"
```

Exploring other fields

Let's have a look at some of the other fields. The headings "`2CreatedM`" and "`20CreatedU`" were for the 3rd and 21st columns respectively. Tell cut to print both those columns as follows:

```
cut -f 3,21 msgraw_sample.txt | less
```

The values in the `CreatedM` column appear to be in ascending time order, so it probably denotes the time the tweet was posted. The `CreatedU` values are out of order, so they might denote the creation time of the User's account on twitter.

What about "`6RT?`" and "`7RTcount`"?

```
cut -f 7,8 msgraw_sample.txt | less
```

Whoa!! That "`RT?`" field is weird. Looks like some programme code.

The second field is mostly zero, which would make sense if it is the count of the number of times the tweet has been retweeted. Most tweets are not very interesting and don't get retweeted by others.

All the values in field 7 looked the same. Let's check for more values by only returning those that don't contain the string "`VAR1`":

```
cut -f 7 msgraw_sample.txt | grep -v "VAR1" | less
```

What do you see? You should see four identical lines, which look like headers. We can check how many different values there are for that field by sorting and count the unique entries:

```
cut -f 7 msgraw_sample.txt | sort | uniq -c
```

Practise 4: What does the "`uniq`" command do exactly, and what is the "`-c`" flag needed for? Did we need to run "`sort`" first? Check by opening the "`man`" (manual) page for "`uniq`":

```
man uniq
```

Type "`q`" to quit the man page.

So the "`RT`"? field seems to be the result of some programming error. But why were there four instances of "`6RT`"? To test if they are header lines, do a simple grep to find all lines containing one of the column names "`2CreatedM`":

```
grep "2CreatedM" msgraw_sample.txt
```

Yes, we've confirmed that the header line is repeated. The wrapped lines are a bit hard to follow though. If you want to see the output without having it wrapped to the size of the

screen, you can use less and give it the "-S" option. You can then use the <left> and <right> arrows to move sideways in the file:

```
grep "2CreatedM" msgraw_sample.txt | less -S
```

Now let's have a look at the location fields: "10Geo#", "11Coord#" and "12Place#"

```
cut -f 11,12,13 msgraw_sample.txt | less
```

Wow! More code. Can anyone guess what programming language is being used? Hint: type in some of the special names "bless" and "undef" in Google to see what you get.

Are there any values for "10Geo#" that is not "undef"?

```
cut -f 11 msgraw_sample.txt | grep -v "undef" | less -S
```

What are these? Are they data structures? What values do they contain? Are they all the same? How many are there?

Let's have a closer look at the values in column 13:

```
cut -f 13 msgraw_sample.txt | grep -v "undef" | less -S
```

What do you think they are? Use the <right> key to see the rest of the line. What about the location field "17Loc$" ?

```
cut -f 18 msgraw_sample.txt | less -S
```

Looks to be location names, but not following any particular vocabulary or consistent format, so probably entered by users manually.

Let's check some examples. Search for "melbourne" using grep:

```
cut -f 18 msgraw_sample.txt | grep -i "melbourne"
```

What does the "-i" flag do? Check by opening the grep man-page:

```
man grep
```

and search (in the man page) for "-i" by typing "/-i".

Are the location entries also geocoded?

```
cut -f 13,18 msgraw_sample.txt | grep -i "melbourne"
```

It doesn't look like they are. No geocoding in that set.
Now explore the additional values like "16Name", "44TZ", etc.

In order to make it easier to identify the right column numbers, you can print out the list of columns names on different lines, by using the "`tr`" command to replace tabs on the first line with newline characters:

```
head -1 msgraw_sample.txt | tr '\t' '\n'
```

So which column is "`Lang`"?

Practise 5: What different languages and time zones are there? Again you can use "`sort | uniq -c`" to organise the output.

Practise 6: Extract the "`Friends`" count, save it to a new file, then read it into R and plot as a histogram. When saving the file, you will need to use "`grep -v`" to remove the duplicated header line (otherwise R will read the data in as a string rather than a number). In R, you will need to cut the data down by removing values bigger than say 3000 before generating the histogram.

Practise 6 is a challenge for your own attempt. This is for you to try them out and discuss among yourselves (on the Forum or on Slack). You can post your shell script solution and R solution on the forum and others can comment on them. We (tutors) will try to remain silent on that unless we think you are on the wrong path. If no one is posting anything, be the first!