Background and motivation for the coursework:

As an information scientist, it is often essential to calculate whether a conversation was successful or not.
For example, you might work for a company that is developing an automated chatbot to interact with customers.
Below is a short sequence of interaction with the system:

Customer: Please could you tell me how to get from A to B?
System: I don't understand what you mean
Customer: Oh come on…please tell me how to get from A to N
System: Sorry I don`t understand.
Customer: Tell me where the ****ing station is

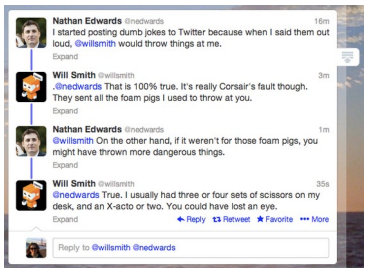
Over five turns, the conversation goes from polite (i.e. positive) to angry (i.e. negative). As a data analyst your task might be to try and determine which strategies by the System lead to negative responses, so that you can avoid them. To do this, you would need to track the level of positivity/negativity and how it *changes during the conversation.*

Similarly, if you were working as a data analyst trying to build a formal model of human relationships, (e.g. if you are a data scientist at facebook, instagram etc. building up a social graph), or if you are a data scientist trying to build an understanding of a criminal network (e.g. you are a data scientist at a government agency, trying to construct a network model of a criminal network)….if you had the following interaction:

A: Hey so how are you?
B: Am good hope you're well
A: So were you able to speak to XYZ
B: No I couldn't man, I am sick of you ordering me around
A: You lazy idiot get out of here
B: *** OFF

(apologies for this very very unimaginative dialogue!)
This interaction also gets progressively worse. If this was an interaction between two criminals in a criminal network, you might judge that their relationship has been weakened, and use that to update your mathematical model of the criminal network. By contrast, if you had an interaction that gets progressively better and more positive, you might judge that the people in the interaction have formed a closer relationship.

Overview:
The purpose of this coursework is to give you experience in monitoring and analyzing live, streaming twitter *conversations*. In this context a "Twitter conversation" is a sequence of tweets where one person responds to another, previous tweet, e.g.



Your goal in this exercise is to write a python (tkinter) program that has **two** frames

- Frame 1 detects and displays live twitter conversations, and saves them to file
- Frame 2 loads twitter conversations from file and performs sentiment analysis on the tweets in order to:
    - identify conversations where the sentiment gets progressively worse (i.e. an argument that escalates)
    - identify conversations where the sentiment gets progressively better

Frame 1:

Write a subclass of a tkinter frame, inside a python app which does the following:

Accessing the twitter feed

- In order to detect twitter conversations you will need to use a wrapper for the twitter API. Depending on what you are familiar with, you could use tweepy (http://www.tweepy.org/) or twython.(https://github.com/ryanmcgrath/twython)
- I strongly recommend creating a new twitter account for this coursework!
- Your app should get its credentials from a text file called "credentials.txt" which contain the relevant authentication details (E.g. key / secret)
- Your app should offer users the possibility of changing the credentials
  (make sure that the GUI can deal with user error)
- Your app should download the live stream of tweets (but only display tweets from conversations – see below)
  The user should have the options in the GUI of specifying
  - the search term(s) (i.e. text, hashtags)
  - the language of the tweets
  - the location of the users.
    - You should let the user type in an address, and it should automatically get the longitude and latitude (for this you can use geopy – https://pypi.python.org/pypi/geopy).
    - Make sure that your app can deal with the error of geopy not finding the longitude/latitude (e.g. simply by disabling it)
    - Allow the user to specify the radius
- If you are using not getting enough tweets from the live stream, then you can also use the .search / .cursor method to search for historical tweets. If you do this, please make sure that your method does not terminate while searching for tweets

Identifying conversations

- To detect whether a tweet is part of a conversation, you can check whether it is responding to a previous tweet. Each tweet has a JSON attribute called "`in_reply_to_status_id`". If this attribute is null/None, it means that it was *not* responding to a previous tweet. If this attribute is non-null, it contains the id of the tweet it was responding to. To get the full conversation you need to get the chain of parent tweets until there are no more parents. You can do this very simply with a recursive function.
- For the purposes of this coursework – your app should only identify and display conversations between 3 and 10 turns.
- Your app should display these twitter conversations in a tree widget.
- Your app should also save these conversations to a file. The search parameters / filters should be saved to the filename.

Part 2: Displaying conversations:

- This frame should have two areas: (1) A window that shows the conversations, and (2) A control panel, where the user can specify criteria for filtering the conversations
- The frame should be able to open the files that have been saved by frame (1) by using a "File" –"Open" menu
- You should offer the user a way of filtering conversational sequences according to their sentiment. For sentiment analysis you can use vader in nltk (it is a very straightforward sentiment analyser). See, e.g. http://www.nltk.org/howto/sentiment.html or http://opensourceforu.com/2016/12/analysing-sentiments-nltk/
- Note that vader gives separate positivity and negativity scores. This means that there are two ways for a conversation to get "worse and worse". Make sure your interface can deal with this.
  (1) Each subsequent turn gets more and more negative
  (2) Each subsequent turn gets less and less positive
  Similarly, there are two different ways of a conversation getting "better and better"
  (1) Each subsequent turn gets less negative
  (2) Each subsequent turn gets more positive
- When the frame is initialized, it should display all the conversations that are in the file
- The control panel should include GUI widgets that allow the user to filter conversations according to specific criteria. (Think for example of flight booking websites where the interface shows sliders, checkboxes and radio buttons to narrow down your search) The conversations that don't match those criteria should NOT be displayed. You can implement it by having an "update" button which, when pressed, refreshes the list of conversations and only displays the conversations that match the criteria specified in the widgets.
- Your app should allow the user to specify the following filter criteria:
  - Minumum number of unique participants
    (should be more than two – anything less isn't conversation)
  - Maximum number of unique participants (up to 10)
  - Minimum length of conversation (in turns)
  - Maximum length of conversation (in turns)
  - Positive and negative sentiment thresholds for *each* of the tweets in the conversation. This should allow the user to easily identify conversations that get progressively worse or that get progressively better or that stay approximately the same.
- The choice of which GUI widgets you use is up to you! Make sure you use suitable widgets.

Deliverables:

- Submit a zip file containing
  - coursework3.py (which reads in credentials from credentials.txt). It should have both your names and student ID at the top of the file
  - a data file of conversations that you have downloaded using frame 1
- Your coursework should be written as a python program with a menu bar. The menu bar should have a file- "exit" option
- Your code must be thread-safe
- Your code must be object oriented and encapsulated. Do not use global variables for variables that are only needed inside a class.
- Your app must use suitable widgets and follow fundamental interaction guidelines.
  The app must be able to deal elegantly with being resized
- You will be graded for both functionality of the code and whether it follows interaction guidelines in the readings and week 5 lecture.