Susie Grimshaw and Riley Chapman
Mining Text from the Web

1. **Project Overview**: We wanted to stream tweets from twitter that contain artist names and run a sentiment analysis on those tweets to determine how the public is feeling about an artist at a given time. We also wanted to create a wordle that showed which words were most frequently used in conjunction with certain artist names. Our approach was to collect data using pattern, create a dictionary that maps each word used in the tweets to the number of times it shows up, and then send our data to wordle.net and create a wordle.

2. **Implementation**: Our code uses lists and dictionaries. The lists were useful for running for loops through all the words in the tweets, and the dictionary was used to keep track of the number of times a word appears in the tweets. Each part (the tweet stream data collection and the creation of the wordle) has an if __main__ = '__name__' statement that allows the script to simply be run. The user simply pastes the text that was copied into the clipboard for them into the first text box they see on the web page that was opened for them. Our code is split into 2 parts, which were implemented in the following ways:

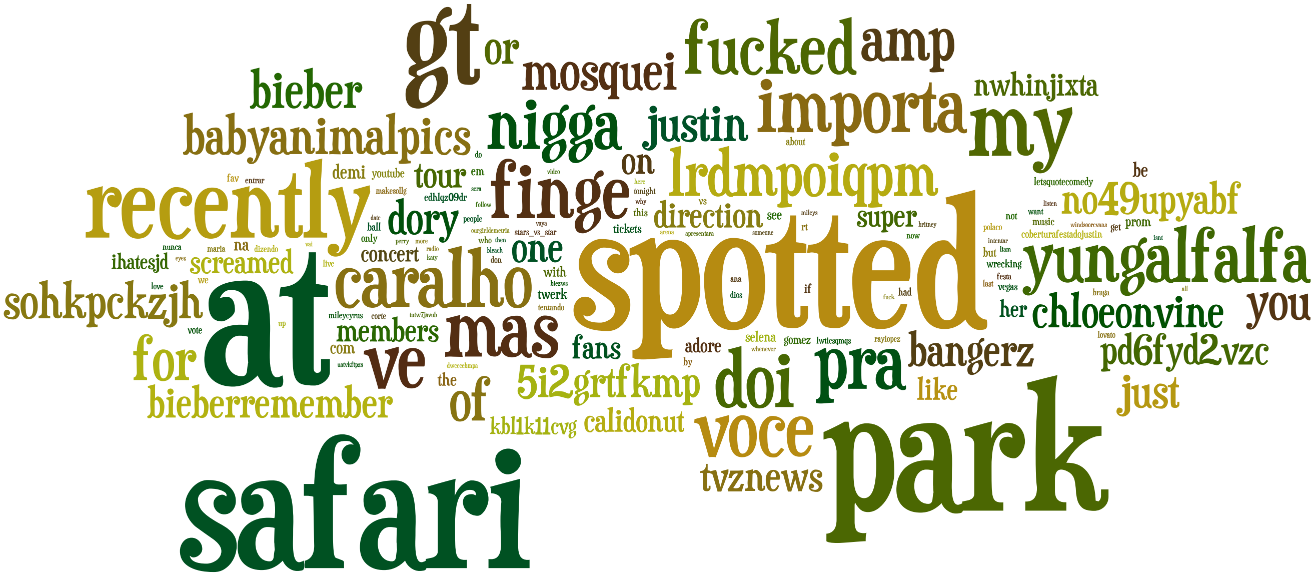
   - **Tweet stream data collection**: We first run the function get_data, that uses the library pattern to collect raw data from twitter. It runs for an inputed amount of time and collects all the tweets that contain an inputed string. It then goes through the function tweets, which pulls out just the text part of the tweet data and returns it as a list of unicode strings (one element for each tweet).

   - **Creation of a wordle**: Function un_uni_list takes in a list of unicode strings and converts it to a list of python strings using the library unicodedata. This list then goes into the function convert_to_LoL_words, which converts this to a list of lists of words using the re library. This then goes through make_lowercase, which makes a single list of all the words as individual strings. Additionally, it converts all uppercase letters to lowercase letters and removes the artist's name in the function remove_artist using operator. Finally, it goes through the function word_frequencies, which returns a dictionary mapping each word in the string to the number of times it is found in the string. We then used the library requests to create a wordle. Since wordle has no API, we need a text input, so make_to_string converts out dictionary into a long string that wordle can recognize as a bunch of words with corresponding weights. We then want to interface directly with the wordle webpage, which happens in wordle_inteface. Originally, we hoped to have the scripts cover all the wordle making for the user, but interfacing with the webpage directly proved to be beyond the scope of this project. Instead, wordle_interface opens the correct web page for the user, as well as copies the correct input for wordle into the clipboard. All that the user must do is paste this text into the readily available text box and click ‘go’.

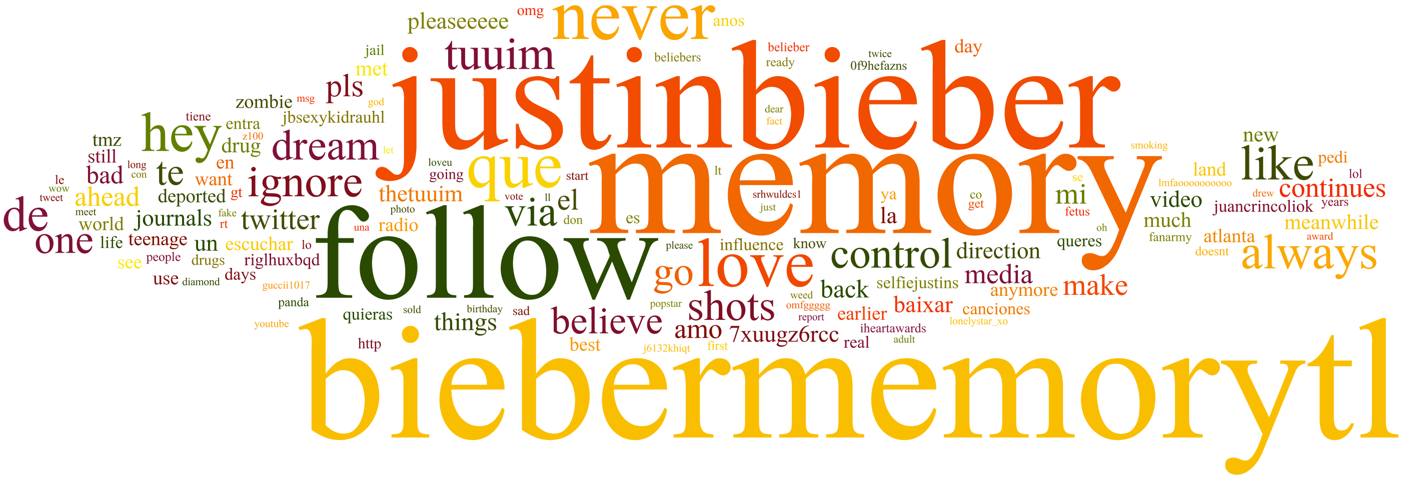
- The libraries used to do this are: webbrowser, pyperclip, operator, re, unicodedata

3. **Results**:

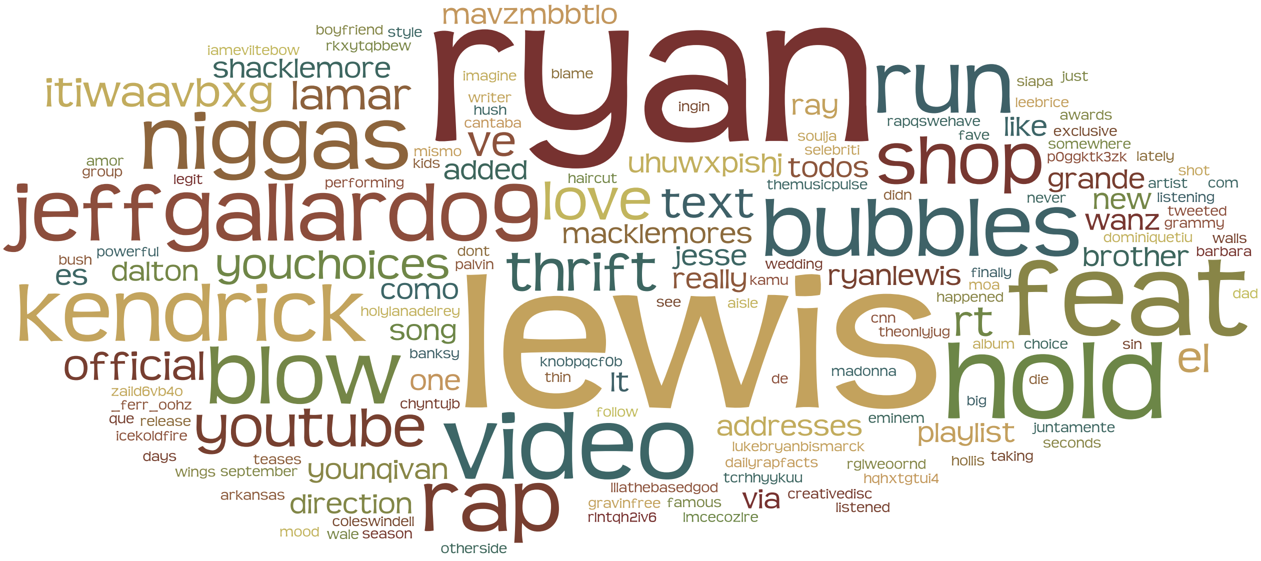
Miley Cyrus



Justin Bieber



Macklemore



4. **Reflection**: We discovered on this assignment that it's considerably harder to merge repositories when both people were changing the code. We found it a lot easier when only one laptop was being used because then we just had one set of changes to add to git rather than two. Similarly, it requires a lot of communication about who is working on what. For example, we found ourselves both writing docstrings for the same function because neither of us had pushed to git in a while. We did scope our project pretty well- it took us 11 hours. Our unit testing system worked pretty well. We would test each function in the command line using a list whose outcome we knew. This prevented us from having problems with functions that are calling other functions.