

Twitter Sentiment Analysis

Software Engineering Project

COMP30440



School of Computer Science and Informatics

University College Dublin

14 15 September 2014

Graham Harkness

05834503

Project Specification

The project is to build a Twitter sentiment analysis program, consisting primarily of two parts:

1. A program which streams, classifies and stores tweets.
2. A web interface which displays the results of the above classification.

The web browser based platform allows logged in users to view a positive and negative tweet classification, most prolific tweeter and average city sentiment derived from tweets of people's opinion of a selection of Irish province capitals, as well as the national feeling about these major population centres.

The end user can log onto the website and read details of the project. The user can navigate through the menu to return information related to the history of sentiment analysis, the details of the sentiment analysis classifier that the project uses and contact details for the project developer. Once provided a username and password, the user can log in and gain access to each city's individual sentiment score. The user can also return the averages of all four cities to return a national sentiment. The user can then log out.

The following is a list of the information the user will have access to:

- The top ten positive *tweeters* in that population center tweeting about that city.
- The top ten negative *tweeters* in that population center tweeting about that city.
- The top ten positive *tweeters* tweeting about that population center globally.
- The top ten positive *tweets* by keyword.
- The top ten negative *tweets* by keyword.
- The total number of *tweets* about a population center.
- The total number of positive versus negative *tweets*.
- The most prolific *tweeter* in a city about that specific city.
- The average sentiment of all *tweeters* about that city this month.

Data will be displayed in a number of forms such as plain text, graphical representation and charts where appropriate. Users will not be able to customise searches and returned results in this iteration of the project. Such features will be included in later versions and are not included in the project specification.

Table of Contents

Table of Contents

Twitter Sentiment Analysis	1
Project Specification	2
1 Introduction	5
1.1 Monitor Opinion Changes	5
1.2 Social Use.....	6
1.3 Citisent	6
2 Design and Implementation	6
2.1 The Web Interface	7
2.2 Sentiment Script	11
2.3 MySQL Database	12
2.4 Results Display	12
3 Results	15
3.1 Speed and Efficiency	15
3.2 Sentiment Trends.....	15
4 System Architecture.....	17
4.1 Website Design (Code)	17
4.2 Script Design (Code).....	24
4.3 Display Results (SQL)	27
4.4 Batch File	32
5 Technologies Used	33
5.1 Python 2.7 (Server Side)	33
5.2 Flask	33
5.3 MySQL	34
5.4 Cron/Task Scheduler.....	34

5.5	Alchemy API	35
5.6	Google Charts / Google Fonts	35
5.7	HTML / CSS	36
5.8	Photoshop	38
6	Problems Encountered.....	39
6.1	Python Compatibility	39
6.2	Twitter API restrictions	40
7	Conclusions	40
8	Future Work	41

1 Introduction

No standardised term has yet been developed to describe the field of the computation and analysis of sentiment, subjectivity and opinion in text (Pang and Lee, 2008). However, a commonly used method, Sentiment Analysis, or Opinion Mining, attempts to address this gap. It provides scientists, researchers and technicians with a computational and analytical technique to study a population's attitudes and opinions towards a given entity (Medhat 2014). This method can be applied to the online social networking site Twitter in an extremely successful way due to Twitter's extremely large cross-sectional reference of people's opinions on a variety of topics in real time.

The value of conducting an analysis of Twitter tweets is evident in a number of sectors. Online consumer reviews and opinions can have serious ramifications for the business environment. For example, the popularity of the online travel review website Tripadvisor has changed the holiday and hotel booking industry dramatically. Research by Nodes (2012) provides strong evidence for this dramatic changes as he showed that 81% of travellers rely on the opinions of other members and their reviews when booking. This strongly suggests that having a handle on the temperature of consumer sentiment at any given time is crucial for those in any sector depending on sales.

Wright (2009) identified Sentiment Analysis as a tool which can be used to return a clear and concise snapshot of consumer emotions and convert it into hard and usable data. While Aspali (2013) argues that the commercial application is the driving factor in the development of sentiment analysis services online, there are additional applications of this data analysis technique beyond this, for example in politics. Needless to say, customer opinion or event voter predictions have been generated without the assistance of an online Sentiment Analysis. However, online Sentiment Analysis has resulted in a lower cost response and a much larger dataset, both in terms of the number of participants and the size of the geographical base. The popularity of this method will only increase in the coming years as its utility becomes clear to invested parties, as illustrated below.

1.1 Monitor Opinion Changes

Sentiment Analysis can be used by businesses in particular to monitor broad consumer changes in customer opinions over time. Surit (2010) supports the technique for its cost efficient method of improving customer service. This method of consumer interaction allows companies to alter or change their customer service approach in response to real time data. Conlin (2013) recommends that companies proactively engage with users posting negative sentiment tweets and attempt to create a positive consumer experience. Sentiment Analysis allows this to be done in real time, efficiently and effectively.

1.2 Social Use

The cost of performing a traditional market research focus group of approximately 8 people averages at about \$6,000 per 90 minute session (Suttle, 2012). Furthermore, this type of qualitative research has the intrinsic limitation of representation (Silverman, 2013). The data generated by this method are restricted as the opinions of one group of people may not be representative of the full range of opinion in that target population. Attempting to gain a wider view of consumer sentiment through phone surveys can cost between \$5000 and \$15000, excluding the report writing and data collection conducted by a market research company.

Politicians rely heavily on market research in order to gauge public opinion (Zarefsky, 1994). This project aims to highlight the utility of city sentiment to social and political planners. It will allow those planners to view changes in sentiment over time and compare those changes to real time situations and news stories. This will ultimately allow for the identification of correlation between city events and population sentiment.

1.3 The Citisent Concept

The concept for this project involves giving people the ability to measure the sentiment in each major city in Ireland on a daily and monthly basis. The concept is in line with ethical and API restrictions and the tweets themselves will only be stored for 24 hours. Tweet metadata can be stored for longer periods, allowing for further analysis in a non-invasive way, including longitudinal studies.

The program measures streams of tweets filtered by the keywords “Galway”, “Dublin”, “Cork” and “Belfast”. The program is designed for use through a web browser interface. It is not for public use at present and is only accessible with a username and password supplied to the user by the site owner. Data are stored on the database are not publically available as per the terms and conditions of the Twitter API.

The program will return data in an easy to understand, user-friendly format. This is a key strength of the program as the program is designed for use by public administration professionals and the users do not require extensive knowledge of code to be able to benefit from the data generated. It allows users to get a real time Sentiment Analysis of how Irish people feel about their major cities and identify trends about these cities that exist over time.

2 Design and Implementation

The project takes the form of two distinct phases: the web interface and the sentiment code. Each is discussed individually below.

2.1 The Web Interface

When developing the concept for the web interface, a number of factors were taken into account. In keeping with the scope of the overall project, three key concepts were taken into account:

- The website must at all times remain lightweight and quick to load. There should be no indulgence in unnecessary design. The website must be clear and concise. The data presentation must be quick to load and easy to interpret.
- The website must remain consistent. It is important that when a user visits pages that the data are presented in a uniform format. Consequently, the data should be easy to compare.
- The website has not been designed for multiple browsers. The website is designed to run primarily 1280 x 1024 pixels. Future work will involve customising the website to work with different device types and screen sizes.

The final website navigation design layout can be seen below in the site wireframes. Key navigation links at the top of the website that require user login access are clearly defined. Website areas that do not require a login are shown on the right of the graph. The login areas provide control of access to the website program while still allowing visiting users to read about the site, sentiment analysis and its benefits.

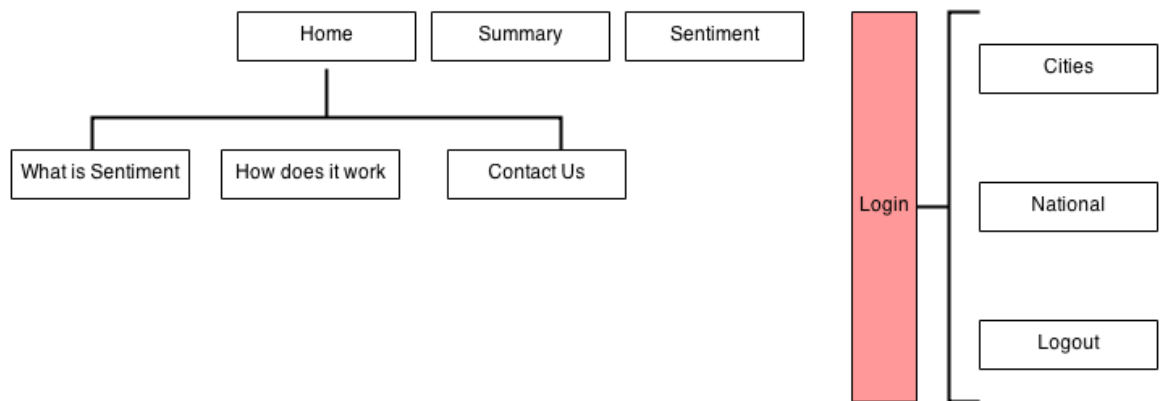


Figure 1 Website Navigation Path

The following diagram shows standard web page navigation within the site. The static pages (left) are clear and concise and return standard website information. The dynamic pages (right) are similar but with more containers which each display dynamic data.

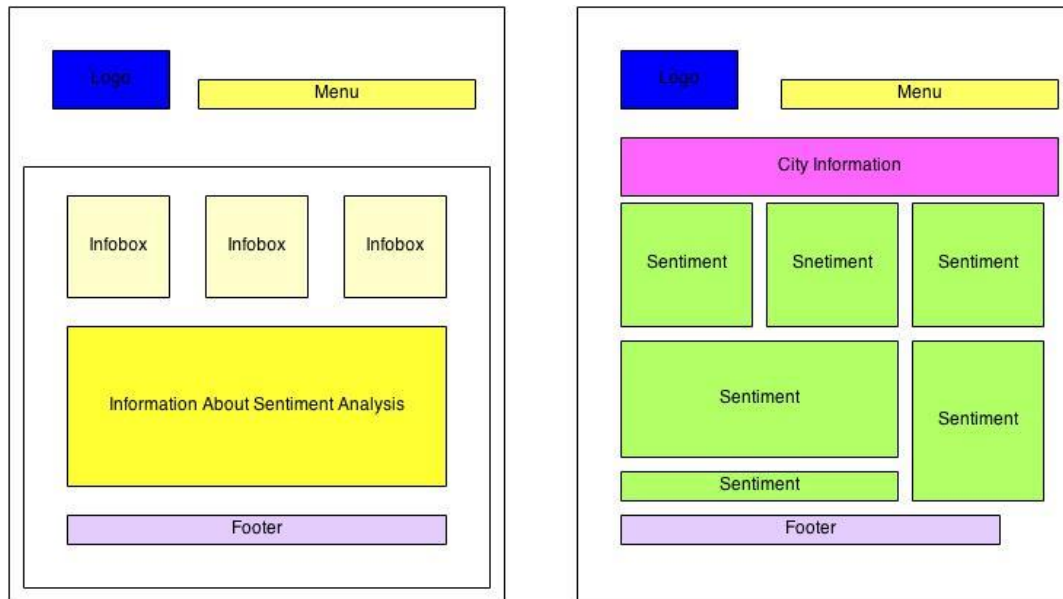


Figure 2 Wireframes of key site pages

Images of the website as it exists can be seen below. Site logo and branding were designed specifically for this project. These images depict an approximation of the final product, complete with the branding and information.



Figure 3 Index.html page which returns links, navbar and summary of the project. This page links to other static pages such as “how it works”, “what is sentiment analysis” and “contact us”

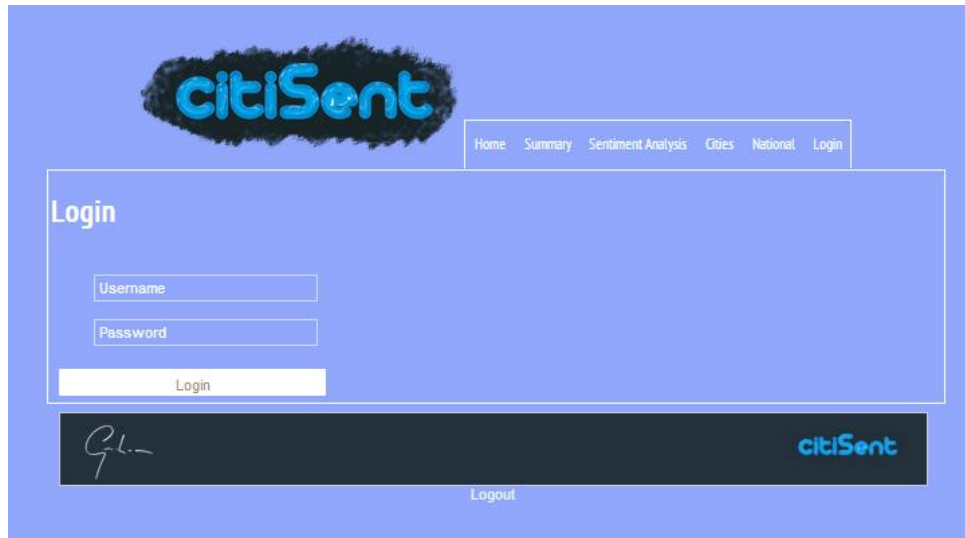


Figure 4 Standard login page to access dynamic site data



Figure 5 Cities.html allows the user to select a location



Figure 6 Dynamic page that queries SQL data and returns the output to the screen formatted in HTML/CSS

2.2 Sentiment Script

The Sentiment Analysis is performed as part of a script which runs on the server, separate from the website. The script runs as a cron using a series of APIs to stream tweets, classify their sentiment and store the tweets in a database. The actual details of the script and APIs will be discussed following an introduction of the underlying concepts.

The program runs a class that takes in all tweets featuring a keyword (ie “Galway”, “Dublin”, “Cork” or “Belfast”) in the string. The class runs on the hour and collects 40 tweets using the keyword. It then stores metadata about the tweet such as username, sentiment, sentiment score ECT. Once 40 tweets have been collected, the script changes keyword to the next city and collects another 40 tweets. This process continues until all cities have 40 tweets. Once that data are stored in a MySQL database it can then be queried by the website.

The script collects a seemingly low number of tweets per hour. The quantity of tweets referring to a specific location differ based on that location and time of day. For example, running the program has shown that people tweet much more about Galway than Cork. It has also been shown that people rarely tweet late at night. In order to ensure a uniform collection of tweets across the cities for further comparisons, the number of tweets collected is limited. Therefore, the same number of tweets per hour per city are collected in order to ensure a non-biased cross section of city sentiment.

Finally, the API used by the program was problematic at first for two reasons. The API is limited to a number of calls. This is currently set at 30,000 tweets (as part of an academic licence). This was problematic as the sentiment classifier returned a 0 sentiment result if this number was exceeded, which in turn produced a skewed sentiment average. Furthermore, if the program was running and the first city did not reach the number of tweets required, the script did not progress to the next city. The program then restarted on the hour collecting tweets of the same city. This happened frequently late at night, when tweets fitting the profile were not being generated.

After consideration of these restrictions created by the API, 40 tweets per hour per city emerged as the most efficient compromise, resulting in 480 tweets per day. This number is further refined based on specific tweet content which will be discussed later in this document. This number ensures

- Broadly similar quantity of tweets collected per city per day
- Standard distribution of tweets chronologically across all cities
- The API limit will not be reached
- Script always ends before it is scheduled to run again

While collecting many more tweets is certainly possible it is not necessary for this project. The collected quantity of tweets has proven effective at representing sentiment adequately. This will be discussed further in the results section of this document.

2.3 MySQL Database

The tweets are stored in a MySQL relational database. The database is called “Twitter” and its structure consists of a table for each city. The tweets are passed to the database and return data for use on the results display. The columns are

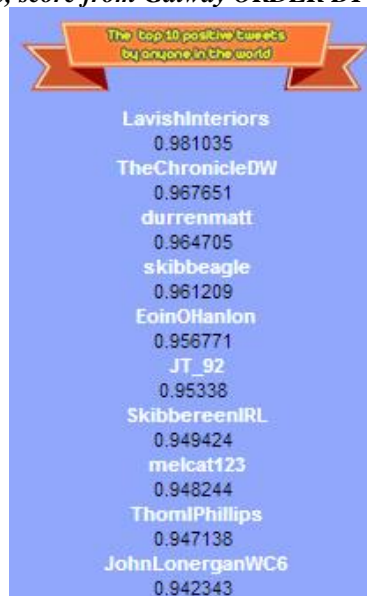
- time – the time and date that the tweet was created
- tweet – stores tweet keywords rather than the tweet itself
- user – stores the username of the tweeter
- location – stores the user defined location of the tweeter
- language – stores the language that the tweet is composed in
- sentiment – stores the overall tweet sentiment (positive / negative / neutral)
- score – stores the classified sentiment of the tweet in a numerical value

The Twitter API terms and conditions state that tweets cannot be stored in a database for longer than 24 hours. As a result, the tweet keywords are displayed on the website as the last 40 tweets from each location. This row is then truncated to *null* every 24 hours to comply with this policy. Tweet metadata can be stored for longer periods. This facilitates the provision of monthly averages of user sentiment and location sentiment ECT.

2.4 Results Display

The results are displayed on a HTML page by querying the SQL database with a series of SELECT statements. The website contains a series of predefined div layouts. Each of these div or container tags displays a different SELECT statement. For example, the following image shows a div tag. At the top of a div tag is a static .png header image which described the results. Below this is a SELECT statement.

SELECT username, score from Galway ORDER BY score DESC limit 10



LavishInteriors	0.981035
TheChronicleDW	0.967651
durrenmatt	0.964705
skibbeagle	0.961209
EoinOHanlon	0.956771
JT_92	0.95338
SkibbereenIRL	0.949424
meicat123	0.948244
ThornlPhillips	0.947138
JohnLonerganWC6	0.942343

Figure 7 Top ten tweets by sentiment

A series of images are also used as a supplementary way to display sentiment. They increase the user friendliness of the program and make the output more attractive and eye-catching. Using programming code to display an image based on a query to the database, the program displays one of the following images:



Figure 8 the images used when assigning a visual aspect to average sentiment

The images maximise the visual element of the site and allow the user to get a quick glimpse of average city sentiment. The use of graphs on the website output is another way in which the visual element of the site is maximised.

The following graph shows an example of the positive negative tweets on a bar chart.



Figure 9 A bar chart showing the number of positive v negative tweets

Finally, the website contains a login feature. This consists of a standard web form with two input fields

- Username
- Password

As this website is for private use only at present, there is no registration page. Username and Password are supplied by the site administrator. There is currently no option for public users to register and access the dynamic data.

3 Results

3.1 Speed and Efficiency

The program streams tweets at a set quantity of 40 per hour per city and inserts the data into a database. A test run of the program in the month of August between the dates Aug 1st and Aug 17th returned a tweet count of 3049. A select statement on all data in this table returned all 3049 rows in 0.01 seconds. The database will always remain relatively small and quick to search. The website returns data almost immediately to ensure a fully displayed infographic with every page load.

Data processing and all SQL statements are performed on the server side, resulting in minimal workload for the client computer. The only requirement for the client computer is access to a compatible browser with HTML/CSS. The program returns data to the server and the same page is returned to all computers who request this HTML data.

In relation to the website, all jpg/png images have been designed with speed of loading as a priority. All images are small and will load quickly.

Two types of tweets are not included in the output. The first are non-subjective tweets. Non-subjective tweets are tweets which return a neutral or no sentiment value. Tweets which do not return a sentiment are not useful. They are therefore collected by the stream but are deleted following a sentiment analysis and are not included in the output. The second are tweets in languages other than English. The API is unable to recognise languages other than English. Tweets which do not contain the language metadata "en" or "en-gb" are collected but are deleted following a Sentiment Analysis and are not included in the output. Ultimately, only usable tweets are stored in the database. As well as maintaining integrity in the results, exclusion of these data minimises waste and increases speed and efficiency.

3.2 Sentiment Trends

The required number of tweets per day to return an adequate sentiment analysis is 384. This is based on using standard sample sizes for required accuracy with an average population between all cities of 308013 people and allowing for a margin of error of 5%. Currently the program collects 480 tweets per city per day. Over time, identifiable trends have emerged within and between the cities.

Results have shown that most tweets which mention the city return an overall positive sentiment. Furthermore, similarities between the numbers of positive versus negative tweets per city have emerged. A 70% positive to 30% negative overall sentiment trend has emerged, independent of city. The same broad trend has emerged for all four cities included in the program in its current form.

City	Pop.
Belfast	280962
Dublin	527612
Cork	191809
Galway	231670
Average	308013

As stated above, with a margin of error of 5% and a 100% response rate an adequate sample size required is 384. Thus negating the need to collect huge numbers of tweets to gain an accurate representation of city sentiment.

Finally, a comment on the validity of Sentiment Analysis is required. Even humans experience difficulty in interpreting statements, despite our extensive experience of sarcasm, humour and conversational nuances. The problem of interpretation is exaggerated when a machine attempts to apply meaning to a statement. The human interpretation of the same statement can be either positive or negative based on the subjective understanding of the phrase (Rhodes 2012). Mullich (2012) gives further examples of where automated Sentiment Analysis techniques can return incorrect data in the form of

- Ironic statements, various forms of humour and other language subtleties and emotions can reverse the tone of a statement. For example, the word “great” in the sentence “*Their customer service really is great! LOL*” suggests a positive interpretation while the “LOL” suffix allows the human reader to understand the tone of the message is sarcastic. A machine would read the message and assign it an overall positive sentiment (Wright, 2009).
- Automated spam filled conversations on twitter which are clearly inauthentic, robotic or wasteful.
- False negatives, where the software sees a negative word but doesn’t comprehend that it is a positive in the overall context of the tweet.
- Cultural & regional differences, where some people from some countries or counties might convey their emotions differently based on colloquialisms.

4 System Architecture

This section will deal primarily with the code used in the creation of the program and website. An overall summary of the individual technology used, including the advantages and disadvantages of the technology and the reasons for choosing specific technology, will be discussed in subsequent sections.

4.1 Website Design (Code)

The website has been designed using Python 2.7. The web framework used for this project was Flask. A web framework is a package or collection of modules which handle the low level Python functions. Flask is a server side technology similar to PHP. Flask is a lightweight framework, also known as a "micro framework", and is based on Werkzeug and Jinja2. Its lightweight implementation design makes it perfect for Citisent.

The architecture of Flask is as follows.

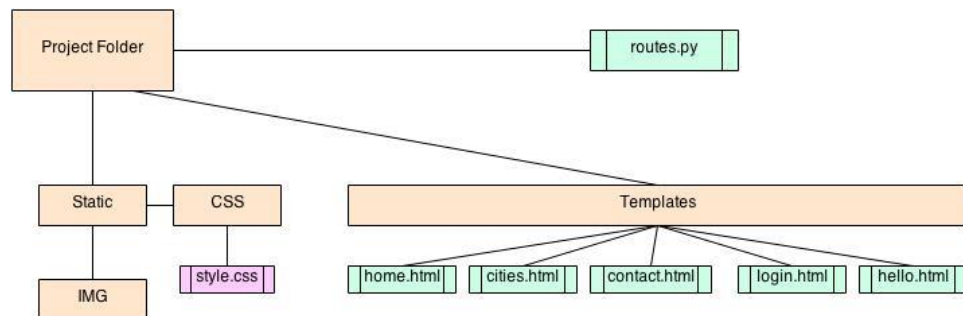


Figure 10 Representation of the site folder for the project

The site structure consists of a static folder, a templates folder and a routes.py folder. The routes.py folder controls communication between the browser and templates. The images and CSS style are housed in the static directory and the html templates are stored in the templates file. The project starts by creating a standard HTML template that can be used as a basis for all other websites. A block content tag is created, which allows the template.html page to inherit from other subpages. See below for the code:

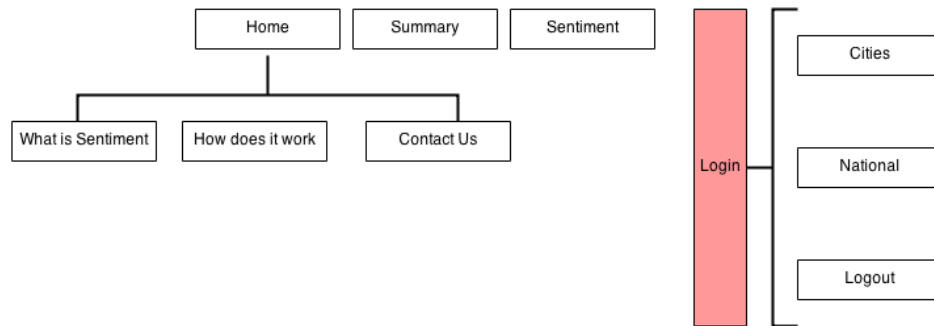


Figure 11 Site Paths

```

<div id=container>
<div id=header>
<div id='navbar'>

<ul>
<li class="active"><a href="/">Home</a></li>
<li><a href="/welcome">Summary</a></li>
<li><a href="/welcome">Sentiment Analysis</a></li>
<li><a href="/cities">Cities</a></li>
<li><a href="/national">National</a></li>
<li><a href="/log">Login</a></li>
</ul>
</div>
</div>
{ % block content % }
{ % endblock % }

```

The framework uses block content tags. These operate similar to PHP. This includes where data are requested to appear within the block tags. This means that code above will remain static across all pages on the site. Then, pages can be included within the block content / endblock tags. One example of this within the Citisent site is the cities.html page. The code can be viewed below:

```

{ % extends "template.html" % }
{ % block content % }
<div id="maincontainer">
  <h1> Pick an Irish City </h1>
  <div class="feature f4">
    <div class="overlay"><a href=" ../dublin">Dublin</a></div>
  </div>
  <div class="feature f5">
    <div class="overlay"><a href=" ../Galway">Galway</div>
  </div>

  <div class="feature f6">
    <div class="overlay"><a href=" ../Cork">Cork</a></div>
  </div>
  <div class="feature f7">
    <div class="overlay"><a href=" ../Belfast">Belfast</a></div>
  </div>
  <p class="marg"> For National statistics please click <a href="national.html">here</a></p>

{ % endblock % }

```

The cities.html file extends the template.html page. The content contained within the

```
{% extends "template.html" %}  
{% block content %}  
  
{% endblock %}
```

Code includes the html/CSS data from the cities.html file into the template.html file to return a compiled html page from the server. CSS is imported from an external style sheet using standard W3C best practice.

The code of a Python based Flask website starts by importing the relevant libraries. In the case of this project, the import code is as follows:

```
from flask import *  
from functools import wraps  
import MySQLDB  
from GChartWrapper import *
```

The modules appear at the top of the routes.py or index.py file. For the purpose of this website the modules included relate to database connectivity and a Python wrapper for Google charts.

Methods used within the website should be defined in this file. For example, the website will need to connect to the database to read the dynamic data. This method is outlined below:

```
def connect_db():  
    db = MySQLdb.connect(host="localhost",  
                        user="graham",  
                        passwd="A05bf311",  
                        db="twitter",  
                        port=3306)  
  
    return db
```

The method connect_db() can be called within any page on the site and can be defined later. A very basic example of a Flask based page path definition can be seen in the index page. In order to render the homepage, its path in the routes.py file must be defined. The index page does not require any dynamic data therefore the code for this page can be seen below:

```
#Render Homepage  
@app.route('/')  
def home():  
    return render_template('home.html')
```

The page route is defined as the route of the site. It has been named "home". When this request to direct to the route directory of the site is made it returns the home.html page to the browser.

Dynamic pages require more definitions to be made within the routes.py file. A comprehensive example of a dynamic page defined in the routes.py file can be seen in the city sentiment page.

```
#Requires login
@login_required
```

```
@app.route('/cork')
def cork():
```

```
#Creates Database Connection
```

```
db = MySQLdb.connect(host="localhost", user="graham", passwd="A05bf311", db="twitter", port=3306)
```

```
#Selects statements from the database are stored in a variable. A variable and select statement are required for every result set to be displayed on this page.
```

```
cur = db.cursor()
cur.execute(
    "SELECT user, score FROM `cork` WHERE location LIKE '%Galway%' OR location LIKE '%Ireland%' ORDER BY `score` ASC LIMIT 10")
tweet = [dict(user=row[0], score=row[1]) for row in cur.fetchall()]

cur.execute(
    "SELECT user, score FROM `cork` WHERE location LIKE '%Galway%' OR location LIKE '%Ireland%' ORDER BY `score` DESC LIMIT 10")
tweets = [dict(user=row[0], score=row[1]) for row in cur.fetchall()]

cur.execute("SELECT user, score FROM `cork` WHERE time LIKE '%Aug%' ORDER BY `score` DESC LIMIT 10")
atweets = [dict(user=row[0], score=row[1]) for row in cur.fetchall()]

cur.execute("SELECT user, tweet FROM `cork` WHERE time LIKE '%Aug%' ORDER BY `score` ASC LIMIT 10")
btweets = [dict(user=row[0], tweet=row[1]) for row in cur.fetchall()]

cur.execute("SELECT user, tweet FROM `cork` WHERE time LIKE '%Aug%' ORDER BY `score` DESC LIMIT 10")
ctweets = [dict(user=row[0], tweet=row[1]) for row in cur.fetchall()]

cur.execute("SELECT `sentiment`,COUNT(`sentiment`)FROM `cork` WHERE time LIKE '%Aug%' GROUP BY `sentiment`")
dtweets = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]

cur.execute("SELECT `sentiment`,COUNT(`sentiment`)FROM `cork` WHERE time LIKE '%Aug%' GROUP BY `sentiment` DESC")
low = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]

low = row[1] /100

cur.execute("SELECT `sentiment`,COUNT(`sentiment`)FROM `cork` WHERE time LIKE '%Aug%' GROUP BY `sentiment` ASC")
high = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]

high = row[1] /100
```

```
#Standard GoogleCharts code stored to variable G
```

```
g= VerticalBarStack([high,low]).title('How Many Tweets').color('orange','blue').label('positive','negative').bar(50,50).size(160,200)
```

```

cur.execute("SELECT `sentiment`,CAST(AVG(`score`) AS DECIMAL(10,2)) FROM `cork` WHERE
time LIKE '%Aug%'")
average = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]
average = row[1]

```

#returns a filename based on query stored in variable "average"

```

if average:
    emotion = 'mosthappy'
elif average:
    emotion = 'superhappy'
elif average:
    emotion = 'mehhappy'
elif average:
    emotion = 'mostsad'
else:
    emotion = 'mostsad'

```

```

cur.execute("SELECT `user`,COUNT(`user`)FROM `cork` WHERE time LIKE '%Aug%' GROUP BY
`user` ORDER BY COUNT(`user`) DESC LIMIT 1")
ftweets = [dict(user=row[0], count=row[1]) for row in cur.fetchall()]

```

#Returns the template but also passes the variables defined here to the html. These variables can then be called in the html page (see below) and their values will be printed on the browser.

```

return render_template('cork.html', tweet=tweet, tweets=tweets, atweets=atweets, btweets=btweets
,ctweets=ctweets, dtweets=dtweets, ftweets=ftweets,g=g, low=low,
high=high,emotion=emotion)

```

Now that the page has been defined in the routes.py file the code contained within the cork.html file can be examined. Content contained within {{ }} tags return the value of the variables passed from the routes.py file. For example {{ tweet }} return the value of the Select statement assigned to the tweet variable.

```

{% extends "template.html" %}
{% block content %}

```

```

<div id="maincontainer">

```

```

    <h1> Cork</h1>
    <p class="marg">Cork City is Ireland's third city (after Dublin and Belfast) and has00_00026.jpg
(25663 bytes)always been an important seaport. It began on an island in the swampy estuary of the River Lee
(the name Corcaigh means a marsh), and gradually climbed up the steep banks on either side.</p>

```

```

<center></center>

```

```

<div class="mainpagecontent" ><center>
    
    {% for item in tweet %}
    <strong><a href="http://www.twitter.com/{{ item.user }}">{{ item.user }}</a></strong><br>{{
item.score }} </br>
    {% endfor %}
</center>

```

```

</div>

<div class="mainpagecontent"><center>


    {% for item in tweets %}
        <strong><a href="http://www.twitter.com/{{ item.user }}">{{ item.user }}</a></strong><br>{{
item.score }} </br>
    {% endfor %}
</center>

</div>

<div class="mainpagecontent">
<center>

    {% for item in atweets %}
        <strong><a href="http://www.twitter.com/{{ item.user }}">{{ item.user }}</a></strong><br>{{
item.score }} </br>
    {% endfor %}
</center>

</div>

<div class="mainpagecontentwide">


    {% for item in ctweets %}
        <strong><a href="http://www.twitter.com/{{ item.user }}">{{ item.user }}</a></strong><br>{{
item.tweet }} </br>
    {% endfor %}
<br>

    {% for item in btweets %}
        <strong><a href="http://www.twitter.com/{{ item.user }}">{{ item.user }}</a></strong><br>{{
item.tweet }} </br>
    {% endfor %}
<br>
</div>

<div class="mainpagecontent">
<center>


    {% for item in dtweets %}
        Of the total number of tweets collected about Cork <strong>{{ item.count }}</strong> were {{
item.sentiment }} </br></br>

    {% endfor %}

    </center>

</div>

<div class="mainpagecontent" style=text-align:justify;>


    {% for item in ftweets %}

```

There was one user who couldnt stop talking about Cork. User **{{ item.user }}** wasp the most prolific tweeter about Cork. They sent messages containing the cities name **{{ item.count }}** times ! You can check out this person's [twitter page](http://www.twitter.com/{{ item.user }}) to find out more.


```
{% endfor %}

#returns image from folder based on number (count) of tweets by user
{% for item in ftweets %}
{% if item.count > 5 and item.count < 9 %}

{% elif item.count > 10 and item.count < 14 %}

{% elif item.count > 15 and item.count < 19 %}

{% elif item.count > 20 and item.count < 50 %}


{% endif %}
{% endfor %}

</div>
<div class="mainpagecontent">
#returns image from folder based overall average sentiment {{ emotion }}




{% endblock %}
```

An example of this data finally rendered can be seen in the screenshot below.

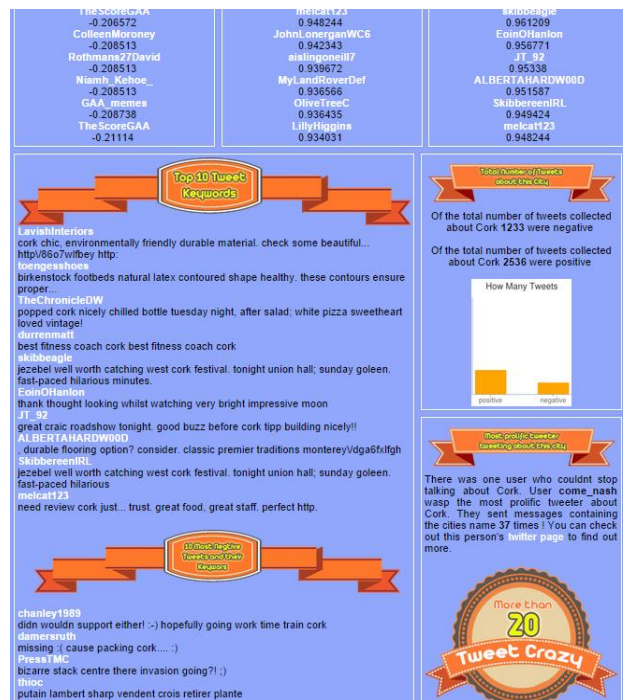


Figure 12 User Experience

4.2 Script Design (Code)

The initial design of this project was to allow a user to enter a keyword search which would, in turn, prompt a script to run collecting and analysing tweets with this search term. This would, in turn, return the results to the user. A number of conceptual issues arose at the planning stage:

- The code would have to run on the client side in order to allow multiple users to run it at once. Python is slower than alternative options for this solution such as Java and C (Lutz 2001) but works well as a server side script. By running the script on the server it means the end user experience will be fast and multi device compatible.
- Issues arose with giving the user the option of entering a search term and quantity of tweets to collect. Uncommon tweeted terms and misspelled words took a long time to return any data to the user or did not return any data at all. When testing initial concepts of the code, terms like “Obama” or “Apple” compiled tweets in under 1 minute and returned the data. Terms such as “Enda Kenny” or “Crumlin” took hours to compile a usable dataset. The end user experience was prioritised in order to minimise the wait time for tweets to be generated and analysed. Forcing a choice from a number of predefined keywords emerged as the most efficient way of compiling a coherent dataset.
- Initially the program had no theme or real world application. Allowing people to enter in a search term and quantity would return data but would provide a theory driven but non-applicable Sentiment Analysis. Using the concept of Citisent to conduct Irish city Sentiment Analysis takes the available technology and APIs to return real world usable data which can be standardised in its output and accessed by anyone, thereby giving the program a real world purpose and making an original contribution to science.

The Sentiment Analysis program follows four distinct phases

1. Stream Tweets
2. Classify and Store Data to Database
3. Run MySQL queries on the data
4. Display the MySQL results within a multidevice browser environment.

The code which performs the sentiment analysis was written in Python 2.7. The advantages and disadvantages of using Python for this project will be discussed in the “Technologies Used” section of this document. A walkthrough of the code is described below. The concept of the sentiment analysis program is that it runs on the server. A crontab (or Windows scheduler when ran locally on the Windows development device) runs a batch file at various intervals. This batch file contains commands to run the sentiment script, Citisent.py. The script starts automatically when the server is powered on and at hourly intervals indefinitely. The programs consists of four classes, called sequentially. Each class 1) streams forty tweets related to a city, 2) analyses the sentiment of the text and 3) stores required data in a MySQL database table which can be 4)

queried by the website. Once the program has completed on the server it will run 60 minutes later. This is in order to ensure that a standard number of tweets is collected across the entire 24 hour period and prevents a bias towards a certain time of day. It also ensures that each class will run and complete within the 60 minute window.

As the program is running a Sentiment Analysis on a city each class uses that city as a keyword when searching tweets. Tweets are collected in real time. The code of the script can be seen below.

```
#Required libraries. Process.py is a library which formats the tweets removing non ASCII data for presentation & rendering purposes only.It does not affect the sentiment API.
```

```
import MySQLDB
import process
import time
```

```
#Tweepy API to Stream Tweets , Alchemy API to classify those Tweets
```

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
from alchemyAPI import AlchemyAPI
```

```
#Required Authentication for Twitter API
```

```
consumerKey = 'yy7WyvomxBu53C0GlnlzHuAE9'
consumerSecret = '2iiG2KElTmRgER2vs8UhlpggXYmmJJUBJqlTceDOYMuhpNjf3X'
accessToken = '24804339-711Szhusk6NWE9oa2uYllH1Tazq1Jjf5zBwuNrcoH'
accessSecret = '36N00STwY038UKBvZ9n5QpJNlkG5vJXt1CbeoWVfcDunv'
```

```
#Database Connection
```

```
db = MySQLdb.connect(host="localhost",
                      user="graham",
                      passwd="A05bf311",
                      db="twitter",
                      port=3306)
```

```
cursor = db.cursor()
```

```
#Classes to Perform Analysis on Tweet Stream
```

```
class corklistener(StreamListener):
class dublinlistener(StreamListener):
class belfastlistener(StreamListener):
class galwaylistener(StreamListener):
```

```
#Calling the Classes & Passing in a Keyword
```

```
auth = OAuthHandler(consumerKey, consumerSecret)
auth.set_access_token(accessToken, accessSecret)
```

```
twitterStream = Stream(auth, galwaylistener())
twitterStream.filter(track=['galway'])
```

```
twitterStream = Stream(auth, dublinlistener())
twitterStream.filter(track=['dublin'])
```

```
twitterStream = Stream(auth, corklistener())
twitterStream.filter(track=['cork'])
```

```
twitterStream = Stream(auth, belfastlistener())
twitterStream.filter(track=['belfast'])
```

Below is an example of one of the classes in the script. Each class is the same where the database Select statement is slightly altered to add to different tables.

```
class galwaylistener(StreamListener):

    def __init__(self, api=None):
        super(galwaylistener, self).__init__()
        self.num_tweets = 0

    def on_data(self, data):

        #Formats the tweets and uses the .split function to store required sections of the tweet stream into
        #variables which will later be stored to the database.
        tweet = process.processTweet(data.split(",text:")[1].split(",source")[0])
        user = data.split("screen_name:")[1].split(",location")[0]
        location = data.split("location:")[1].split(",url")[0]
        language = data.split("lang:")[1].split(",contributors_enabled")[0]

        # Creates an instance of the API class and uses the predefined .sentiment method to return a
        #score stored in the response variable.
        alchemyAPI = AlchemyAPI()
        response = alchemyAPI.sentiment("text", tweet)
        self.num_tweets = self.num_tweets + 1

        #Printing for Error Checking
        print self.num_tweets
        print tweet

        #Try for Tweets with No Sentiment Type (Prevents Crashes). If no sentiment found return char
        #“Unknown”. Basic Error Handling.
        try:
            t = response["docSentiment"]["type"]
        except:
            t = "Unknown"

        #Try for Tweets with No Sentiment Score (Prevents Crashes) If no sentiment found return int “0”. Basic
        #Error Handling
        try:
            s = response["docSentiment"]["score"]
        except:
            s = 0

        #Insert the data into the database
        cursor = db.cursor()
        cursor.execute("INSERT INTO galway(time, tweet, user, location, language, sentiment, score)
VALUES (%s,%s,%s,%s,%s,%s,%s)",
        (time.asctime(time.localtime(time.time())), tweet, user, location, language, t , s ))

        #Remove Unknown or Neutral (objective & unusable tweets) from the database. Also remove non English
        #tweets from the database.
        cursor.execute("DELETE FROM Galway WHERE sentiment='Unknown' OR sentiment='neutral';")
        cursor.execute("DELETE FROM tweets WHERE language!='en' or language!='en-gb;")
        db.commit()

        #Ends the instance of the class when 40 tweets have been collected
        if (self.num_tweets >= 40):
            return False
```

```

else:
    return True

def on_event(self, status):
    print status

```

4.3 Display Results (SQL)

Once the data have been stored in the database, the program employs a number of methods to demonstrate how data can be displayed in a format that is both easy to understand and usable. Citisent returns a standard webpage with dynamic data.

- SQL

Plain text is returned using a series of SQL queries.

```
"SELECT user, score FROM `dublin` WHERE location LIKE '%Dublin%' OR location LIKE '%Ireland%'
ORDER BY `score` ASC LIMIT 10"
```

Positive Tweets by Irish People:

Returns the username and sentiment score from a city where the location of the tweet has been defined within the tweet as Dublin or Ireland. This includes tweets about Dublin by Dublin people or Irish people ignoring non Irish locations. It then displays the top 10 by sentiment score. Tweets streamed and analysed for the month of August 2014 are currently used for these tweets.

```
"SELECT user, score FROM `dublin` WHERE location LIKE '%Dublin%' OR location LIKE '%Ireland%'
ORDER BY `score` DESC LIMIT 10"
```

Negative Tweets by Irish People:

Returns the username and sentiment score from a city where the location of the tweet has been defined within the tweet as Dublin or Ireland. This includes tweets about Dublin by Dublin people or Irish people ignoring non Irish locations. It then displays the bottom 10 by sentiment score. The month of August is used for these tweets.

```
SELECT user, score FROM `dublin` WHERE time LIKE '%Aug%' ORDER BY `score` DESC LIMIT 10
```

Positive Tweets by anyone in the World:

Returns all tweets globally that use the keyword of the city and return the top ten scored by sentiment. Uses data from the month of August

```
SELECT user, tweet FROM `dublin` WHERE time LIKE '%Aug%' ORDER BY `score` ASC LIMIT 10
```

Negative Tweets by anyone in the World:

Returns all tweets globally that use the keyword of the city and return the bottom ten scored by sentiment. Uses data from the month of August.

```
SELECT user, tweet FROM `dublin` WHERE time LIKE '%Aug%' ORDER BY `score` DESC LIMIT 10
```

```
SELECT `sentiment`, COUNT(`sentiment`)FROM `dublin` WHERE time LIKE '%Aug%' GROUP BY `sentiment`"
```

```
SELECT `sentiment`, COUNT(`sentiment`)FROM `dublin` WHERE time LIKE '%Aug%' GROUP BY `sentiment` DESC"
```

```
SELECT `sentiment`, COUNT(`sentiment`)FROM `dublin` WHERE time LIKE '%Aug%' GROUP BY `sentiment` ASC"
```

The above returns counts of tweets which can be used to return data resulting in the total number of tweets using this city name as a keyword.

```
SELECT `user`,COUNT(`user`)FROM `dublin` WHERE time LIKE '%Aug%' GROUP BY `user` ORDER BY COUNT(`user`) DESC LIMIT 1
```

This statement returns the user who tweeted the most with this city as a keyword in their tweets. The data above are displayed in the website as plain text. The tweets have been processed to return only keywords and lowercased data to improve readability.

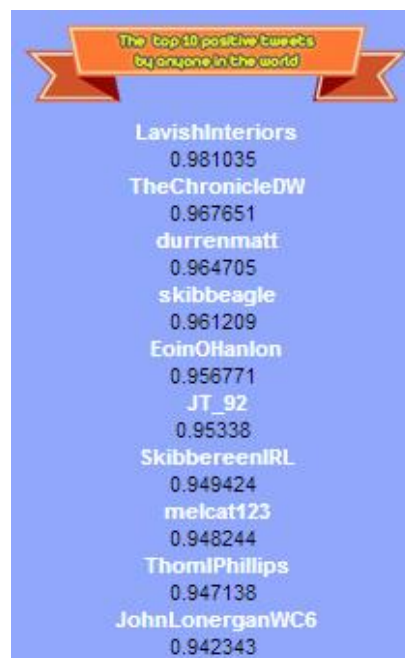


Figure 13 Example of Output to Browser

- Images

Images have been stored in the directory of the website. Based on the data returned by the sentiment score, an “if else” statement is ran and this selects an image. This is to improve the user experience and make the data easier to interpret. The program starts by running a Select statement on the score data to get the total average and then return it to two decimal places. Then, based on the data returned, assigned a char (which correlates with the name of an image file in the img folder without the file extension) and assigned the variable “emotion” the name value which is the name of the file. The file is then displayed on the website accordingly.

```
SELECT CAST(AVG(`score`) AS DECIMAL(10,2)) FROM `dublin` WHERE time LIKE '%Aug%'
average = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]
average = row[1]
```

```
if average > 0.30 and average < 0.50:
    emotion = 'mosthappy'
elif average > 0.25 and average < 0.29:
    emotion = 'superhappy'
elif average > 0.10 and average < 0.24:
    emotion = 'mehhappy'
elif average > 0 and average < 0.9:
    emotion = 'mostsad'
else:
    emotion = 'angry'
```



Figure 14 Example of Output to Browser

This can also be achieved by use of the “for loop” which is also used.

```
{% for item in ftweets %}
```

```
    There was one user who couldnt stop talking about Cork. User <strong>{{ item.user }}</strong> wasp  
    the most prolific tweeter about Cork. They sent messages containing the cities name <strong>{{ item.count  
    }}</strong> times ! You can check out this person's
```

```
    <a href="http://www.twitter.com/{{ item.user }}">twitter page</a> to find out more. </br></br>
```

```
    {% endfor %}
```

```
{% for item in ftweets %}
```

```
{% if item.count > 5 and item.count < 9 %}
```

```

```

```
{% elif item.count > 10 and item.count < 14 %}
```

```

```

```
{% elif item.count > 15 and item.count < 19 %}
```

```

```

```
{% elif item.count > 20 and item.count < 50 %}
```

```

```

```
{% endif %}
```

```
{% endfor %}
```

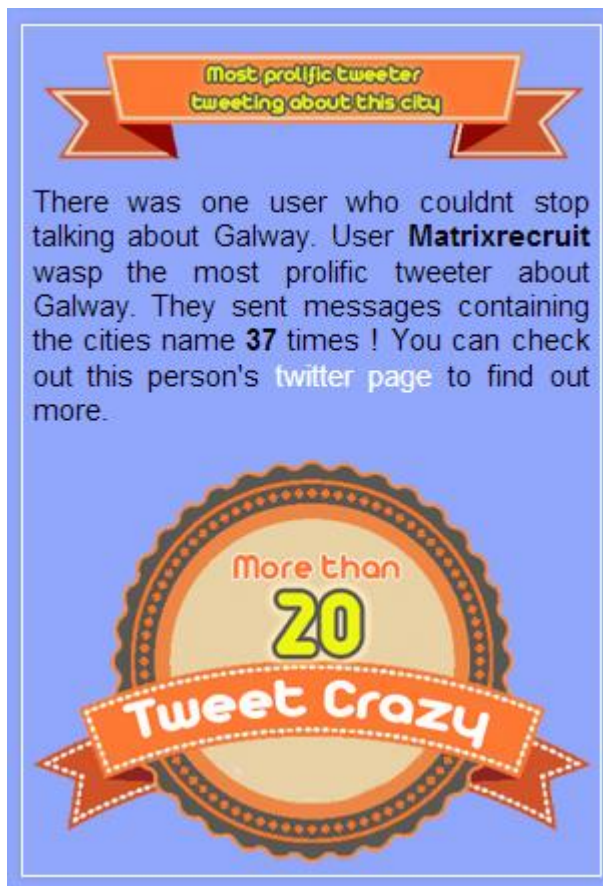


Figure 15 Example of Output to Browser

- Google Charts

The Google chart API was also used to demonstrate how data can be formatted into a chart and displayed online. Using the GChartwrapper to display the data and SQL statements to get the total number of positive and negative tweets about a city, those data can be passed to the histogram. Google Charts then displays the data in an easy to understand format.

```
cur.execute("SELECT `sentiment`, COUNT(`sentiment`)FROM `dublin` WHERE time LIKE '%Aug%'
GROUP BY `sentiment` DESC")
low = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]

low = row[1] /100

cur.execute("SELECT `sentiment`, COUNT(`sentiment`)FROM `dublin` WHERE time LIKE '%Aug%'
GROUP BY `sentiment` ASC")
high = [dict(sentiment=row[0], count=row[1]) for row in cur.fetchall()]

high = row[1] /100

g= VerticalBarStack([high,low]).title('How Many Tweets').color('orange','blue').label('positive',
'negative').bar(50,50).size(160,200)
```



Figure 16 Example of Output to Browser

4.4 Batch File

As outlined previously, the program is designed to be ran automatically at predefined intervals. Development and hosting is currently on a device running Windows OS. Development of the project involves the use of a *.bat file, which runs the script. A Windows task scheduler runs the *.bat file at hourly intervals. The file contents can be viewed below.

```
@echo off
:start
cls
echo Press Key To Run

set Pathname = C:\Users\graha_000\Desktop\College\Flask\
cd %Pathname%

start python citisent.py
```


5 Technologies Used

5.1 Python 2.7 (Server Side)

Choosing Python as the language of the project represents an initiative on the part of the developer to experiment with a new language and gain experience in methods beyond those included on the curriculum of the MSC Conversion course.

The Python script runs on the server compiling data and sending it to a MySQL database. The choice of running the script on the server was made for the following reasons:

- As the script does not need to run on the client side, the program ensures that all devices with a web browser will have access to the program. The program is not reliant on the client having installed a Python interpreter on their device. The program is compatible with mobile devices and tablets as well as desktop computers.
- As the script is ran independently of a device, the program will constantly run on the server storing information. The end user will not have to wait when querying data or for a program to finish running. Consequently, the user experience will be seamless and when the page is reloaded the most recent data will be automatically displayed.
- While Python runs slower than other programs such as Java (Van Rossum, 1997), it requires less code. As the speed of the program for this project is ran on the server at a set time, the speed is not an issue in the same way it would be on a client device. The program therefore saves space by using a language that requires less code to implement.
- Python code is very easy to read and interpret. It is easy to write, read and understand. There are also several libraries and compatible APIs for Python that can be implemented in this project and future expansions of the project.

5.2 Flask

Flask is a web framework for Python based on the Werkzeug and Jinja2 templates. A web framework is a collection of packages which allow Web applications to be written without having to handle low level details such as protocols or sockets. Flask is a server side technology. Similar to Python, it is not reliant on complex programs on the client device. The client need only have a compatible web browser installed.

The use of Flask represents an extension of the initiative undertaken by the developer. The website could have been completed in PHP but it was deemed an interesting exercise to use a Python based web framework for web design on the basis of increasing experience and expertise.

Flask is open source and is a lightweight method of creating websites which mix html/css code with the programming concepts of control statements such as a “for loops” and “if/else statements”, both of which were employed in this project.

Flask was very easy to set up and integrated well with MySQL.

5.3 MySQL

MySQL is a popular Relational Database Management system. Websites such as Facebook, Wikipedia all use MySQL. Furthermore, it integrates well with Python and Python based web frameworks. There are a number of advantages with using MySQL as a database system.

- Open Source

MySQL is free and open source. As a result, most servers support MySQL. Alternatives for this project were MySQLite. This is designed for lightweight mobile applications but was unavailable on any servers. The second alternative was MongoDB, another popular database solution. Developer experience with MySQL and its high level of server support eventually emerged as the deciding factors in favour of MySQL.

- Easy

This project does not require a complex relation database structure. A small number of tables and a series of select and insert statements were required. There is no PL/SQL required so MySQL is a perfect solution.

- Fast

The database queried all results from Dublin for the month of August (4751 entries) in 0.01 seconds. As a result MySQL is sufficiently fast for the purposes of this project.

- Cross Platform Support

MySQL runs on a variety of platforms including Linux and Windows. MySQL also has extensions provided for it by many common languages including Python, PHP, C++. The open source nature of the database allows APIs for most commonly used programming languages.

- Memory Efficient

MySQL prevents memory leaks and is an efficient database solution. A single database can hold 8TB of data if required. In the case of this project only relatively small amounts of plain text are being stored. Size requirements are considered low at this stage however MySQL provides for future scalability.

5.4 Cron/Task Scheduler

Cron is a Linux utility which schedules a command or script to run on the server automatically at a specified time and date. The Cron was used to automate the repetitive task of running the script hourly over the course of a month.

The script runs and collects 40 tweets per city per hour. The program stops and then is ran again automatically at the beginning of the next hour. This ensures that a sample of tweets are collected efficiently and that an accurate average sentiment is maintained.

As the development computer for this project was a Windows device the Cron is run as part of the Windows Task Scheduler program.

5.5 Alchemy API

The sentiment score of a tweet is obtained by use of the Alchemy API. Alchemy uses machine learning to perform natural language processing on bodies of text. The API provides a free version which limits the number of calls to 1000 per day. This is a sufficient number for the program in its current form. However, the developer holds an academic license from Alchemy, which allows 30k calls per day. While this is not necessary for the project at this stage, it does allow the project to be greatly expanded over many multiple locations.

Alchemy API returns both a numeric score and a text based sentiment (positive, negative, neutral) for each tweet that is passed to it. Once this information is stored to a database it can be organised and queried.

Tweepy API is a Python based library that is used to facilitate the streaming of tweets. It enables Python to communicate with the Twitter platform and use its API. Tweepy accesses Twitter by using OAuth authentication which is required by Twitter.

```
from tweepy import Stream
from tweepy import OAuthHandler
from tweepy.streaming import StreamListener
```

As part of the program, the entire content of the tweet, including metadata, is streamed and normalised into required variables for use later within the program. This eliminates the need to stream separate data. Tweepy uses the method `on_data()` as part of the `StreamListener` function to listen for data when data are generated. A `filter()` method is passed to `StreamListener` to restrict the listener to search for terms of a specific keyword.

5.6 Google Charts / Google Fonts

Google Charts is a free tool that allows data to be passed to the API in the form of a variable and return those data as an infographic. In order to ensure that Google Charts will work with Python, the wrapper `GChartWrapper` must be imported to the program.

Google Fonts allows web developers to use non-system fonts on a website and to ensure that the design of the website will be retained across multiple devices without any requirement that the font is installed on the device itself. Google Fonts is a free service and provides access to fonts that are hosted on Google servers.

5.7 HTML / CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are two standard core technologies for building websites. The structure of the page is delivered by the HTML and the design & aesthetic elements are provided for by the linked CSS file. All pages in this website are .html extensions. The Flask framework allows dynamic data to be used on a html page contrasted to PHP which required a PHP extension.

CSS is linked externally to the HTML pages with some minor exceptions on individual images and DIV tags. The CSS file is called style.css and an example of its content can be seen below.

```
        text-decoration: none;
        color: #fff;
    }
    a img {
        outline : none;
    }
    h1, h2, h3 {
        font-family: 'PT Sans Narrow', sans-serif;
        color:white;
    }

    p{
        font-family: 'PT Sans Narrow', sans-serif;
        text-align:justify;
        color:black;
    }

    p.marg{

        margin-right: 150px;
        margin-left: 50px;

    }

    .image {
        margin: 5px 5px 5px 5px;
    }
    body {
        margin-right: auto;
        margin-left: auto;
        background-color: #90a7fc;

    }
    #header {
        text-align:center;
        height: 150px;
        width: 100%;
        background-color:;
        margin-right: auto;
        margin-left: auto;
        position: relative;
        border-top-style: none;
        border-right-style: none;
        border-bottom-style: none;
```

```

        border-left-style: none;
        max-width:3000px;
        margin-bottom: 0px;
        border-top:10px;
    }

    #container {
        max-width:1200px
        margin-right: auto;
        margin-left: auto

    }

    #navbar {
        height: auto;
        width: 100%;
        margin-right: auto;
        margin-left: auto;
        text-align: center;
        position: relative;
        border-top-style: none;
        border-right-style: none;
        border-bottom-style: none;
        border-left-style: none;
        margin-top: 60px;
        max-width:1000px;
    }

    #maincontainer {
        height: auto;
        width: 200%;
        margin-right: auto;
        margin-left: auto;
        max-width:1000px;
        position: relative;
        border-top-width: 1px;
        border-right-width: 1px;
        border-bottom-width: 1px;
        border-left-width: 1px;
        border-top-style: solid;
        border-right-style: solid;
        border-bottom-style: solid;
        border-left-style: solid;
        border-top-color: #fff;
        border-right-color: #fff;
        border-bottom-color: #fff;
        border-left-color: #fff;
        padding-left:2px;
        padding-right:2px;
    }

    .mainpagecontent {
        float: left;
        margin: 5px;
        position: relative;
        border: thin solid #fff;
        box-shadow: 0 0 5px rgba(0, 0, 0, 0.15);
        /* height: 200px; */
        width: 310;
        padding: 5;
        height: auto;
    }

```

```

.mainpagecontentwide
{
float: left;
margin: 5px;
position: relative;
border: thin solid #fff;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.15);
/* height: 200px; */
width: 63%;
padding: 5;
height: auto;
}

.feature {
float: left;
margin: 5px;
position: relative;
border: thin solid #fff;
box-shadow: 0 0 5px rgba(0, 0, 0, 0.15);
}
.f1 {
background-image: url(/static/img/BG-5.jpg);
height: 200px;
width: 320px;
}

```

5.8 Photoshop

Adobe Photoshop CC is considered to be the industry standard in photo editing and photo manipulation. It is an excellent tool for the creation of images for both website and app image development. Images for this website have been altered and manipulated in Photoshop to fit the overall design and colour scheme of the project.

Employing Photoshop allowed the program to integrate into a themed user experience in particular with regards to the logo design and image design. All images are licenced for public use.

6 Problems Encountered

6.1 Python Compatibility

The deployment strategy for this project was to host the entire program on the UCD School of Computer Science and Informatics server. Issues relating to compatibility emerged early on. The UCD server was running an earlier version of Python (Version 2.6). The project was created using code and libraries compatible with the most commonly used version of Python, version 2.7. This necessitated an upgrade of the Python version available in the School of Computer Science and Informatics in UCD.

Further problems presented themselves when running both the sentiment script and web framework on the server. Modules installed were not available to use without granting root privileges on the server itself. Unless the user was logged in as an administrator Python would not recognise the modules as installed. This affected modules that were installed with both Flask and the sentiment script. Multiple support sessions with the School of Computer Science and Informatics in UCD were unable to find a solution to this issue and were unable to grant root privileges. As a result the entire project was hosted locally on the development machine at submission time.

Furthermore, efforts to host the project on a premium hosting package offered by Hosting Ireland failed due to the fact that they did not provide Python 2.7 on their server.

Hi Graham,

On the current server (and this would include the newest as well), it has CentOS 6, which uses version 2.6.6. If you specifically need 2.7 then it would require a VPS of your own to use this version.

Regards,

Pat Cremin

The final option was to set up an apache fileserver that this author had administrator rights to in order host the project for demonstration purposes and access. This allows full control over the versions of all required modules and languages.

6.2 Twitter API restrictions

This project runs by collecting data from a large number of tweets daily and processing the sentiment of each tweet. Currently the programs collects

Number of Cities	Tweets Per Hour	Tweets Per City Per Day	Total Tweets Per Month
4	160	3840	115200

Unfortunately the free Alchemy API sentiment analysis library would only process a maximum of 1000 tweets per day. This meant that the free version of Alchemy would not be sufficient for this project nor would the project be scalable. Furthermore once the 1000 API calls per day are met, Alchemy would return 0 or Neutral for all subsequent calls. As a result SQL queries which calculate averages over time would be inaccurate.

This restriction also influenced the project design as the designed involved scraping a defined number of tweets per hour rather than a program allowing the user to enter a number of tweets to search. Earlier tests showed that creating a program that allowed a user to search a keyword and 500 tweets meant that the program could only be ran twice within any 24 hour.

In order to overcome these limitations, this project required the use of the Alchemy academic licence which allows a key to pass 30k tweets per day and return accurate sentiment analysis. In its current form the program is sufficiently supported by the free licence. It can be significant expanded within the confines of the academic license. However, should the project be expanded beyond Irish cities, for example, it may be necessary to purchase the commercial licence and scale to include many more cities.

7 Conclusions

Sentiment Analysis

The program is a lightweight, server side website designed to deliver an enjoyable user experience. The output is sentiment data, produced within a predefined context and presented in an easy to understand and concise format. The key strengths of the program are that the final user experience is quick to load, very user-friendly and is compatible with any device with a web browser installed. The predefined variables, ie keywords “Galway”, “Dublin”, “Cork” and “Belfast”, and quantity of tweets collected, ie 40, mean that the program will act predictably and within its scope for every user. The control of the key components of the program and its variables is retained by the developer as an efficient and effective way of maintaining a smooth and uninterrupted user experience.

8 Future Work

The key future expansion of this project is to expand it to include many cities across multiple countries. The structure of the data storage and delivery means that free to use software such as Google Translate can be utilised to stream tweets by language. This opens up infinite possibilities for expansion to other locations and languages. Furthermore, additional features to the website are planned:

- Allowing users to select a month or day from a drop down menu and displaying results from historical data. The program has been running since 1st August, therefore it will take more time to build up a store of historical data.
- Allowing the user to search for specific users and return a history of that user's sentiment score.
- An increase in the complexity of the SQL queries would allow for more in-depth examination of data trends.
- Expanding the SQL database would allow for direct comparison between cities, a feature not currently offered by the program.

References

- Aspali, A. (2013). *Who benefits from Sentiment Analysis?* Retrieved on 30th June 2014 from <http://socialmediatoday.com/aspilalleli/1180346/who-benefits-sentiment-analysis>.
- Conlin, B. (2013). *The benefits of Sentiment Analysis: A marketer's guide*. Retrieved on 30th June 2014 from <http://www.vocus.com/blog/the-complete-guide-sentiment-analysis-for-marketers/>.
- Medhat, W., Hassan, A. & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 30(10), 2-20.
- Mullich, J. (2012). *Improving the effectiveness of customer Sentiment Analysis*. Retrieved on 30th June 2014 from <http://data-informed.com/improving-effectiveness-of-customer-sentiment-analysis/>.
- Nodes, S. (2012). *Underscoring the importance of online hotel reputation management*. Retrieved on 30th June 2014 from <http://www.tnooz.com/article/underscoring-the-importance-of-online-hotel-reputation-management/>.
- Pang, B & Lee, I. (2008). Opinion Mining and Sentiment Analysis. *Foundations and Trends in Information Retrieval*, 2(1-2), 1–135
- Rhodes, M. (2010). *The problem with automated Sentiment Analysis*. Retrieved on 30th June 2014 from <http://www.freshminds.net/2010/05/the-problem-with-automated-sentiment-analysis/>.
- Silverman, D. (2013). *Doing Qualitative Research: A Practical Handbook*. London: Sage Publications Ltd.
- Suttle, R. (2011). *Key costs & benefits in marketing research*. Retrieved on 30th June 2014 from <http://smallbusiness.chron.com/key-costs-benefits-marketing-research-26311.html>.
- Surit, A. (2013). *How can Sentiment Analysis benefit your business?* Retrieved on 30th June 2014 from http://shoutanalytics.blogspot.ie/2013/02/how-can-sentiment-analysis-benefit-your_7.html.
- Wright, A. (2009). *Mining the web for feelings, not facts*. Retrieved on 30th June 2014 from http://www.nytimes.com/2009/08/24/technology/internet/24emotion.html?_r=2&.

Additional Resources

<http://blogs.dirteam.com/blogs/sanderberkouwera/archive/2011/10/14/dcpromo-advanced-mode-what-does-it-do.aspx>.

<http://oreilly.com/catalog/python2/chapter/ch15.html>.

<https://www.python.org/doc/essays/comparisons/>.

<http://www.pythoncentral.io/introduction-to-tweepy-twitter-for-python/>.

<http://quod.lib.umich.edu/j/jala/2629860.0015.204/--public-sentiment-is-everything-lincolns-view-of-political?rgn=main;view=fulltext>.

http://sharpened.com/web_design_rules.

[http://technet.microsoft.com/en-us/library/cc738121\(v=ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc738121(v=ws.10).aspx).

https://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/choosing_between_ntfs_fat_and_fat32.mspx?mfr=true.

<http://www.w3.org/standards/webdesign/htmlcss>.

<http://www.washington.edu/itconnect/learn/workshops/online-tutorials/graphics-and-design-workshops/adobe-cs/photoshop/>.