

# Trends in tweets

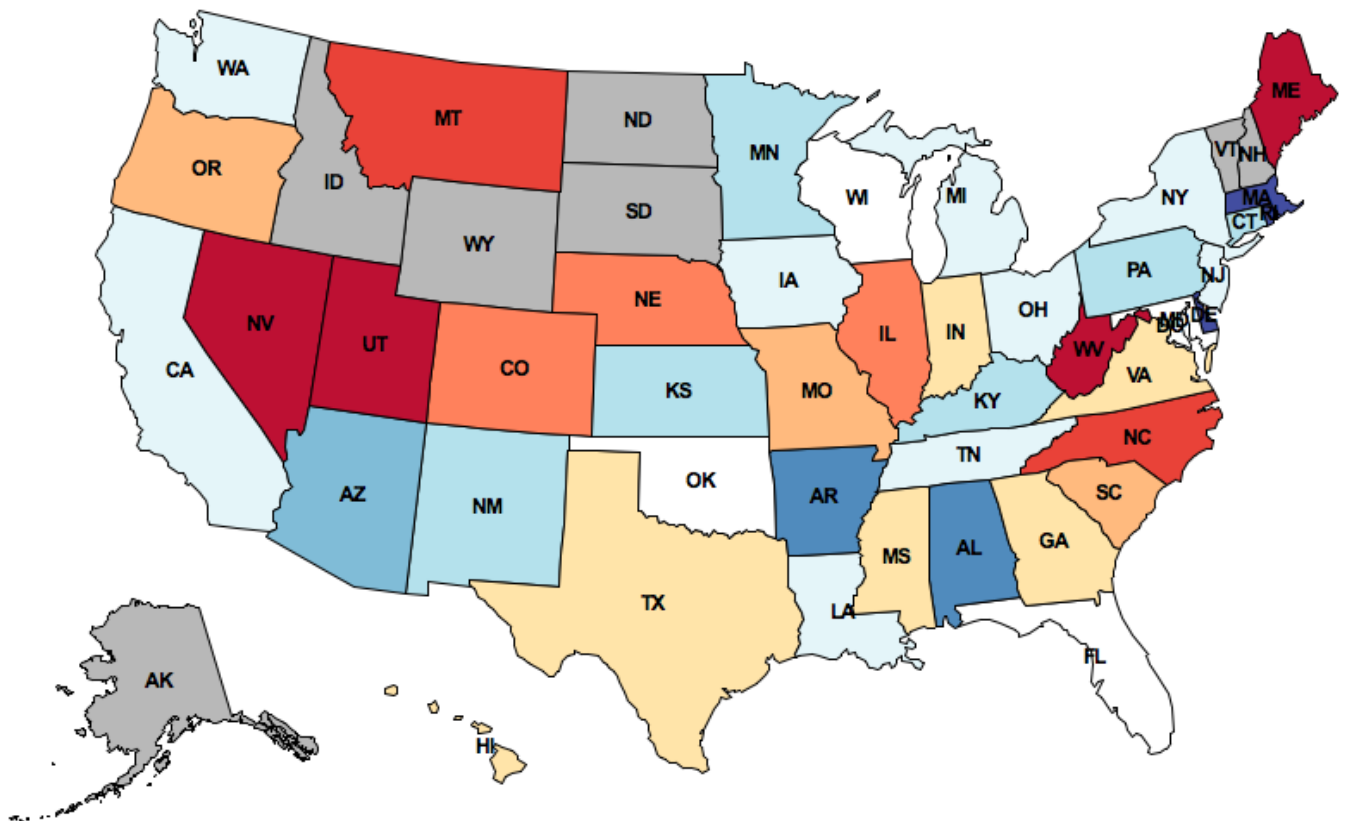
AUTHOR  
Christian Osendorfer

PUBLISHED  
October 31, 2023

## Introduction

**Acknowledgements:** This project is taken from the excellent [CS61A course in Berkeley](#).

In this project, you will develop a geographic visualization of twitter (nowadays called **x**) data across the USA. You will need to use dictionaries, lists, and data abstraction techniques to create a modular program.



The map displayed above depicts how the people in different states feel about Texas. This image is generated by:

1. Collecting public Twitter posts (tweets) that have been tagged with geographic locations and filtering for those that contain the "texas" query term,
2. Assigning a sentiment (positive or negative) to each tweet, based on all of the words it contains,
3. Aggregating tweets by the state with the closest geographic center, and finally
4. Coloring each state according to the aggregate sentiment of its tweets. Red means positive sentiment; blue means negative.

The details of how to conduct each of these steps is contained within the project description. By the end of this project, you will be able to map the sentiment of any word or phrase.

You will get the necessary starter code (`proj01.zip`) and two versions of the dataset: The small version of the dataset (as part of `proj01.zip` in the `data` folder) is enough to work through all of this project. The full dataset (`full_data.zip`) can be used by you if you have enough time: You need to copy the unzipped file into the projects `data` folder.

The project uses several files, but all of your changes will be made to the first one (`trends.py`).

- `trends.py`: A starter implementation of the main project file.
- `geo.py`: Geographic positions, 2-D projection equations, and geographic distance functions.
- `maps.py`: Functions for drawing maps.
- `data.py`: Functions for loading Twitter data from files.
- `graphics.py`: A simple Python graphics library.
- `ucb.py`: Utility functions.

#### Tip

The only file you should need to change is `trends.py`.

## Logistics

---

This is a multi-week project (yet, you *should be able* to finish it within 1 week). You'll work in a team of up to three people, and you can complete all problems together with your teammates. However, you need to be able to answer **all potential questions** during the oral *Leistungsnachweis* **by your own**.

Start early! Feel free to ask for help early and often on moodle!

#### Important

The various questions are not too difficult, yet require some thinking and additional work (e.g. taking a formula from wikipedia and putting it into a python expression). To get used to this type of challenges, **start early**.

## The Feelings in Tweets

In this section you will create an abstract data type for tweets, split the text of a tweet into words, and calculate the amount of positive or negative feeling in a tweet.

## Tweets

---

First, you will implement an abstract data type for Tweets. The constructor `make_tweet` is defined at the top of `trends.py`. `make_tweet` returns a python dictionary with the following entries:

text: a string, the text of the tweet, all in lowercase  
time: a datetime object, when the tweet was posted  
latitude: a floating-point number, the latitude of the tweet's location  
longitude: a floating-point number, the longitude of the tweet's location

**Question 1.** Implement the `tweet_words` and `tweet_time` selectors. Call `extract_words` to list the words in the text of a tweet.

**Question 2.** Implement the `tweet_location` selector, which returns a position. Positions are another abstract data type, defined at the top of `geo.py`. Make sure that you understand how to manipulate positions; they play an important role in this project.

When you complete questions 1 and 2, the doctest for `make_tweet` should pass:

```
python3 trends.py -t make_tweet # try to find out what this `-t` is!
```

**Question 3.** Improve the `extract_words` function as follows: Assume that a word is any consecutive substring of text that consists only of ASCII letters. The string `ascii_letters` in the string module contains all letters in the ASCII character set. The `extract_words` function should list all such words in order and nothing else.

When you complete this problem, the doctest for `extract_words` should pass.

```
python3 trends.py -t extract_words
```

**Question 4.** Implement the `sentiment` abstract data type, which represents a sentiment value that may or may not exist. The constructor `make_sentiment` takes either a numeric value within the interval -1 to 1, or `None` to indicate that the value does not exist. Implement the selectors `has_sentiment` and `sentiment_value` as well. You may use any representation you choose, but the rest of your program should not depend on this representation.

When you complete this problem, the doctests for `make_sentiment` and `get_word_sentiment` should pass. You can also call the `print_sentiment` function to print the sentiment values of all sentiment-carrying words in a line of text.

```
python3 trends.py -t make_sentiment
python3 trends.py -t get_word_sentiment
python3 trends.py -p computer science is my favorite!
python3 trends.py -p life without lambda: awful or awesome?
```

**Question 5.** Implement `analyze_tweet_sentiment`, which takes a tweet (of the abstract data type) and returns a sentiment. Read the docstrings for `get_word_sentiment` and `analyze_tweet_sentiment` to understand how the two functions interact. Your implementation should not depend on the representation of a sentiment!

When you complete this problem, the doctests for `analyze_tweet_sentiment` should pass.

```
python3 trends.py -t analyze_tweet_sentiment
```

# The Geometry of Maps

## Positions

---

We will use the `position` abstract data type to represent geographic latitude-longitude positions on the Earth. The data abstraction, defined at the top of `geo.py`, has the *constructor* `make_position` and the *selectors* `latitude` and `longitude`.

In this section, you will write two functions that together determine the centers of U.S. states. The shape of a state is represented as a list of polygons. Some states (e.g. Hawaii) consist of multiple polygons, but most states (e.g. Colorado) consist of only one polygon (still represented as a length-one list).

**Question 6.** Implement `find_centroid`, which takes a polygon and returns three values: the coordinates of its centroid and its area. The input polygon is represented as a list of `position` abstract data types, which are the *consecutive vertices* of its perimeter. The first vertex is **always identical** to the last.

The centroid of a two-dimensional shape is its center of balance, defined as the intersection of all straight lines that evenly divide the shape into equal-area halves. `find_centroid` returns the centroid and area of an individual polygon.

The formula for computing the [centroid of a polygon](#) appears on Wikipedia. The formula relies on vertices being consecutive (either clockwise or counterclockwise; both give the same answer), a property that you may assume always holds for the input.

When you complete this problem, the doctest for `find_centroid` should pass.

```
python3 trends.py -t find_centroid
```

**Question 7.** Implement `find_center`, which takes a shape represented by a list of polygons and returns a position, its centroid.

The centroid of a collection of polygons can be computed by [geometric decomposition](#). The centroid of a shape is the weighted average of the centroids of its component polygons, weighted by their area.

When you complete this problem, the doctest for `find_center` should pass.

```
python3 trends.py -t find_center
```

Once you are finished, `--draw_centered_map` (or `-d` for short) as an argument to `trends.py` will draw the 10 states closest to a given state (including that state).

```
python3 trends.py -d CA
```

## The Mood of the Nation

# States

---

The name `us_states` (in `geo.py`) is bound to a dictionary containing the shape of each U.S. state, keyed by its two-letter postal code. You can use the keys of this dictionary to iterate over all the U.S. states.

In this section, you will write functions to determine the state that a tweet is coming from, group tweets by state, and calculate the average positive or negative feeling in all the tweets associated with a state.

**Question 8.** Implement `find_closest_state`, which returns the two-letter postal code of the state that is closest to the location of a tweet. Use the `geo_distance` function (provided in `geo.py`) to calculate the shortest distance in miles between two positions.

When you complete this problem, the doctests for `find_closest_state` should pass.

```
python3 trends.py -t find_closest_state
```

**Question 9.** Implement `group_tweets_by_state`, which takes a list of tweets and returns a dictionary. The keys of the returned dictionary are state names (two-letter postal codes), and the values are lists of tweets that appear closer to that state's center than any other.

When you complete this problem, the doctests for `group_tweets_by_state` should pass.

```
python3 trends.py -t group_tweets_by_state
```

**Question 10.** As an exercise, implement `most_talkative_state`, which returns the state containing the most tweets containing a given term.

When you complete this problem, the doctests for `most_talkative_state` should pass.

```
python3 trends.py -t most_talkative_state
```

**Question 11.** Implement `average_sentiments`. This function takes the dictionary returned by `group_tweets_by_state` and also returns a dictionary. The keys of the returned dictionary are the state names (two-letter postal codes), and the values are average sentiment values for all the tweets in that state.

If a state has no tweets with sentiment values, leave it out of the returned dictionary entirely. Do not include a state with no sentiment using a zero sentiment value. *Zero represents neutral sentiment*, not unknown sentiment. States with unknown sentiment will appear gray, while states with neutral sentiment will appear white.

You should now be able to draw maps that are colored by sentiment corresponding to tweets that contain a given term.

```
python3 trends.py -m sandwich
python3 trends.py -m obama
```

```
python3 trends.py -m texas
python3 trends.py -m my life
```

If you downloaded the small version of the project, you will only be able to map these four terms. If you would like to map any term (assuming it is in the large data set), you will need to download this Twitter data file and place it in the data directory of your project.

## Into the Fourth Dimension

In this last section, you will take into account the fourth dimension: time. Each tweet has a `datetime` object, which represents the time that the tweet was posted.

The provided `draw_map_by_hour` function visualizes the tweets that were posted during each hour of the day. For example, “sandwich” tweets appear most positive at `10:00pm`: late night snack!

**Question 12.** Implement `group_tweets_by_hour`, which takes a list of tweets and returns a dictionary. The keys of the returned dictionary are the integers 0 through 23, representing the 24 hours of the day. The value for each key is a list of the tweets that were posted during that hour.

To get started, read the online [documentation](#) on datetime objects. When you are finished, you can visualize the changing sandwich sentiments over time.

```
python3 trends.py -b sandwich
```

### Important

Congratulations 🎉! You finished your first python programming project 100!

## Extensions

These extensions are optional (and hence are not graded). If you have time left and enjoy programming for fun, try some of these (or your own ideas!). However, please do not change the behavior or signature of the functions you have already implemented or are provided in the repo.

- Punctuation can be an indicator of sentiment as well. Add an emoticon (smiley) detector that attributes positive sentiment to happy faces :-)) and negative sentiment to sad ones.
- In the standard implementation, some tweets are associated with different states than the ones in which they occurred. For example, all tweets from Manhattan are assigned to New Jersey. New Yorkers would be appalled! Write a function `find_containing_state` that finds the state that actually contains a tweet position.
- The `graphics.py` package supports animation. Use the `slide_shape` method to have states and dots slide into place.
- Correct the spelling of tweets before you compute their sentiment.