Poniższy rozdział pochodzi z książki:

Charles R. Severance - "Python dla wszystkich: Odkrywanie danych z Python 3"

Pełna wersja podręcznika znajduje się na stronie https://py4e.pl/book

Rozdział 16

Wizualizacja danych

Nauczyliśmy się pewnych aspektów Pythona, sprawdzaliśmy, jak go używać oraz jak korzystać z sieci i baz danych w celu zarządzania danymi.

W poniższym rozdziale przyjrzymy się trzem kompletnym aplikacjom, które łączą to wszystko, żeby zarządzać danymi i je wizualizować. Możesz później użyć tych aplikacji jako przykładowego kodu, który pomoże Ci rozpocząć rozwiązywanie Twojego problemu.

Każda z tych aplikacji jest w pliku ZIP, który możesz pobrać na swój komputer, rozpakować i uruchomić.

16.1. Budowanie mapy OpenStreetMap z geokodowanych danych

W tym projekcie wykorzystamy API geokodowania OpenStreetMap (usługa Nominatim), po to by zamienić wpisane przez użytkowników nazwy uczelni na lokalizacje geograficzne[1], a następnie umieścimy przetworzone dane na mapie OpenStreetMap.

Aby rozpocząć, pobierz aplikację z poniższego adresu:

https://py4e.pl/code3/geodata.zip

W warunkach korzystania z usługi Nominatim jest wskazanie, by ograniczyć się do maksymalnie jednego zapytania na sekundę (usługa jest darmowa, stąd też gdybyśmy generowali bardzo dużą liczbę zapytań w krótkim czasie, to prawdopodobnie szybko zablokowano by nam dostęp do API). Nasze zadanie dzielimy na dwie fazy.

W pierwszej fazie bierzemy nasze dane wejściowe z pliku `where.data` i odczytujemy je wiersz po wierszu, pobieramy zgeokodowaną odpowiedź serwera Nominatim i zapisujemy ją w bazie danych (plik `geodata.sqlite`). Zanim użyjemy API geokodowania, sprawdzamy, czy w naszej bazie ("pamięci podręcznej") mamy już dane dla tego konkretnego wiersza, dzięki czemu w przypadku ponownego uruchomienia programu nie będziemy musieli drugi razy wysyłać zapytania do API.

W dowolnym momencie możesz uruchomić cały proces od początku, wystarczy usunąć wygenerowany plik `geodata.sqlite`.

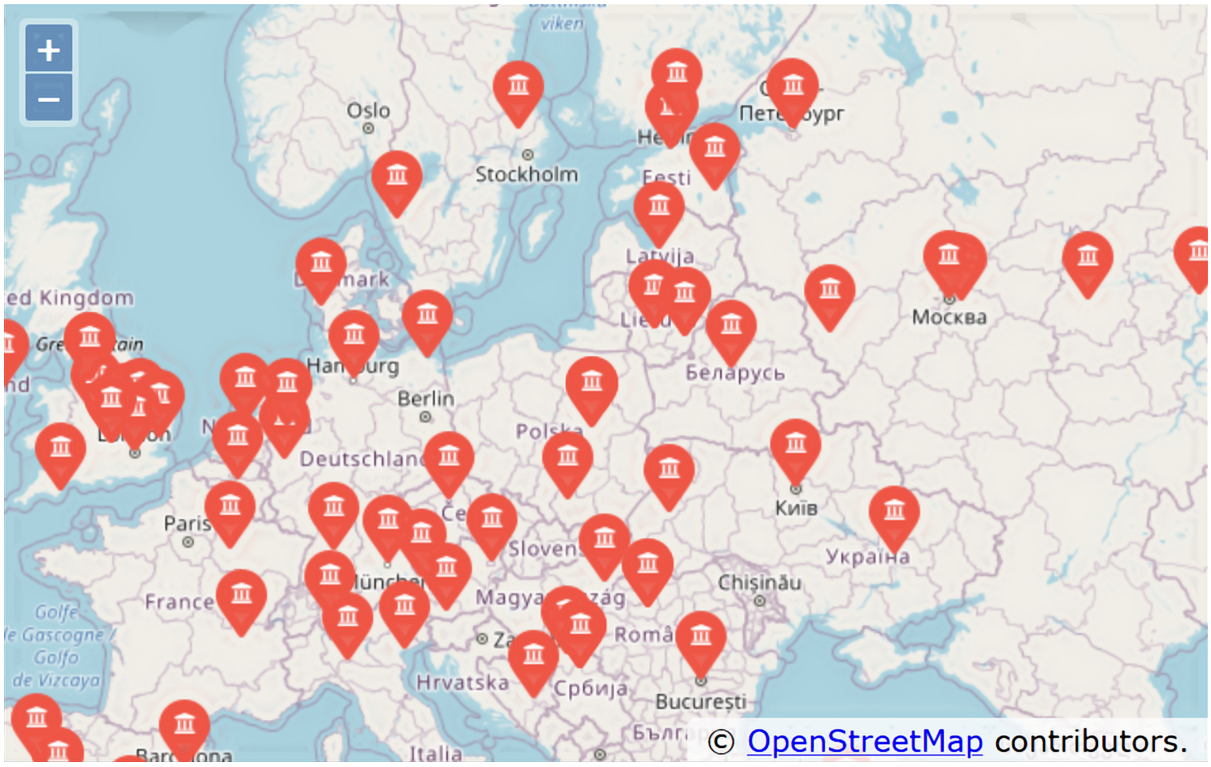
Uruchom program `geoload.py`. Program ten odczyta wiersze wejściowe z pliku `where.data` i dla każdego wiersza sprawdzi, czy jest on już w bazie danych. Jeśli nie mamy jeszcze danych dla przetwarzanej lokalizacji, to wywoła on zapytanie API geokodowania, aby pobrać dane i zapisać je w bazie SQLite.

Oto przykładowe uruchomienie, kiedy w bazie danych znajdują się już jakieś dane:

```
Znaleziono w bazie  AGH University of Science and Technology

Znaleziono w bazie  Academy of Fine Arts Warsaw Poland
```

[1] Należy mieć na uwadze, że taka zamiana tekstu na dane geolokalizacyjne nie zawsze daje precyzyjne i poprawne wyniki, co będzie można zauważyć w części wyników tej aplikacji.



Rysunek 16.1. Mapa OpenStreetMap

```
Znaleziono w bazie  American University in Cairo

Znaleziono w bazie  Arizona State University

Znaleziono w bazie  Athens Information Technology

Pobieranie https://nominatim.openstreetmap.org/search.php?q=...
Pobrano 954 znaków {"type":"FeatureColl

Pobieranie https://nominatim.openstreetmap.org/search.php?q=...
Pobrano 822 znaków {"type":"FeatureColl

...
```

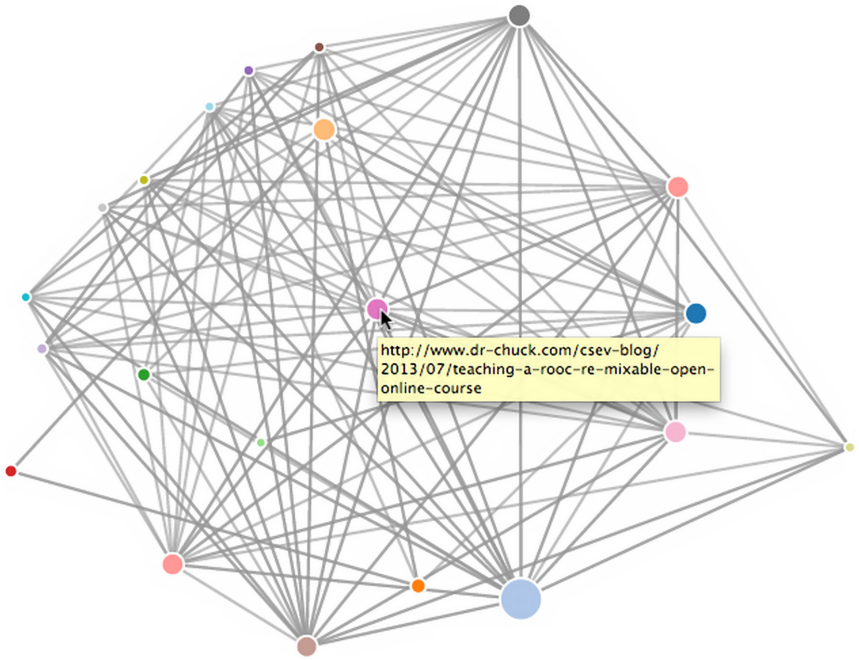
Pierwsze pięć lokalizacji znajduje się już w bazie danych, więc są one pomijane. Program działa tak do momentu, w którym znajdzie niezapisane lokalizacje, i wtedy zaczyna o nie odpytywać API.

Plik `geoload.py` może zostać zatrzymany w dowolnym momencie, a ponadto kod zawiera licznik (zmienna `count`), którego można użyć do ograniczenia liczby połączeń do API geokodowania w danym uruchomieniu programu.

Po załadowaniu danych do `geodata.sqlite`, możesz je zwizualizować za pomocą programu `geodump.py` ↪ . Program ten odczytuje bazę danych i zapisuje plik `where.js` zawierający lokalizacje, szerokości i długości geograficzne w postaci wykonywalnego kodu JavaScript. Pobrany przez Ciebie plik ZIP zawiera już wygenerowany `where.js`, ale możesz go wygenerować jeszcze raz, aby sprawdzić działanie programu `geodump.py`.

Uruchomienie programu `geodump.py` zwraca następujący wynik:

```
Akademia Górniczo-Hutnicza ... Polska 50.065703299999996 19.918958667058632
Akademia Sztuk Pięknych ... Polska 52.2397515 21.015564130658333
...
260 wierszy zapisano do where.js
Otwórz w przeglądarce internetowej plik where.html, aby obejrzeć dane.
```



Rysunek 16.2. Algorytm PageRank

Plik `where.html` składa się z kodu HTML i JavaScript, które służą do wizualizacji mapy OpenStreetMap przy pomocy biblioteki OpenLayers. Strona odczytuje najświeższe dane z pliku `where.js` po to, by uzyskać dane niezbędne do wizualizacji. Oto format pliku `where.js`:

```
myData = [
[50.065703299999996,19.918958667058632, 'Akademia Górniczo-Hutnicza, ... ,
    ↪ Polska'],
[52.2397515,21.015564130658333, 'Akademia Sztuk Pięknych, ... , Polska'],
    ...
];
```

Jest to lista list zapisana w języku JavaScript. Jej składnia jest bardzo podobna do składni Pythona.

By zobaczyć lokalizacje na mapie, otwórz plik `where.html` w przeglądarce internetowej. Możesz najechać kursorem na każdą pinezkę na mapie i kliknąć, żeby znaleźć lokalizację, którą zwróciło API kodowania dla danych wejściowych wprowadzonych przez użytkownika. Jeżeli po otwarciu pliku `where.html` nie widzisz żadnych danych, sprawdź, czy w przeglądarce jest włączony JavaScript lub czy są jakieś błędy w konsoli deweloperskiej swojej przeglądarki.

16.2. Wizualizacja sieci i połączeń

W poniższej aplikacji będziemy wykonywać niektóre funkcje znajdujące się w mechanizmie wyszukiwarki internetowej. Najpierw przeczeszemy mały fragment sieci internetowej, uruchomimy uproszczoną wersję algorytmu Google PageRank, by określić strony, do których jest najwięcej odniesień z innych stron (czyli by wskazać, które strony są potencjalnie najistotniejsze), a następnie zwizualizujemy rangę strony i połączenia w naszym małym zakątku sieci. Aby tworzyć wizualizacje, wykorzystamy bibliotekę JavaScript D3[2].

Możesz pobrać i rozpakować poniższą aplikację:

https://py4e.pl/code3/pagerank.zip

Pierwszy program (`spider.py`) wczytuje stronę internetową i wprowadza serię stron do bazy danych (`spider.sqlite`), rejestrując przy tym odnośniki (linki) pomiędzy stronami. W każdej chwili możesz zrestartować

[2] https://d3js.org/

cały proces, usuwając plik `spider.sqlite` i uruchamiając ponownie `spider.py`.

```
Wpisz adres internetowy lub wciśnij Enter:
['https://www.dr-chuck.com']
Ile stron: 25
1 https://www.dr-chuck.com (8386) 4
4 https://www.dr-chuck.com/dr-chuck/resume/index.htm (1855) 9
12 https://www.dr-chuck.com/dr-chuck/resume/pictures/index.htm (1827) 5
...
Ile stron:
```

W powyższym przykładowym uruchomieniu wskazaliśmy naszemu robotowi, by sprawdził domyślną stronę i pobrał 25 stron. Jeśli zrestartujesz program i wskażesz, by przeszukał więcej stron, to nie będzie on ponownie przeszukiwał stron już znajdujących się w bazie danych. Po ponownym uruchomieniu robot przechodzi do losowej, niesprawdzonej jeszcze strony i zaczyna tam swoją pracę. Tak więc każde kolejne uruchomienie `spider.py` dodaje tylko nowe strony.

```
Wpisz adres internetowy lub wciśnij Enter:
Kontynuowanie istniejącego indeksowania stron. Usuń spider.sqlite,
aby rozpocząć nowe indeksowanie.
['https://www.dr-chuck.com']
Ile stron: 3
22 https://www.dr-chuck.com/csev-blog/category/uncategorized (91096) 61
27 https://www.dr-chuck.com/csev-blog/2014/09/how-... (59001) 20
30 https://www.dr-chuck.com/csev-blog/2020/08/styling-... (33522) 21
Ile stron:
```

Możesz mieć wiele punktów startowych w tej samej bazie danych – w programie nazywane są one "witrynami". Jako następną do sprawdzenia robot internetowy wybiera losową stronę spośród wszystkich nieodwiedzonych linków na wszystkich witrynach.

Jeżeli chcesz wyświetlić zawartość bazy `spider.sqlite`, możesz uruchomić `spdump.py`:

```
(16, None, 1.0, 2, 'https://www.dr-chuck.com/csev-blog')
(15, None, 1.0, 30, 'https://www.dr-chuck.com/csev-blog/2020/08/...')
(15, None, 1.0, 22, 'https://www.dr-chuck.com/csev-blog/category/...')
...
22 wierszy.
```

Dla danej strony program pokazuje liczbę przychodzących linków, stary współczynnik PageRank, nowy współczynnik PageRank, id strony oraz adres URL. Program `spdump.py` pokazuje tylko te strony, które mają co najmniej jedno odniesienie z innych stron.

Gdy będziesz już mieć w swojej bazie danych kilka stron, to za pomocą programu `sprank.py` możesz uruchomić algorytm obliczania współczynnika PageRank. Musisz tylko podać, ile iteracji algorytmu program ma wykonać.

```
Ile iteracji: 2
1 0.720326643053916
2 0.34992366601870745
[(1, 0.6196280991735535), (2, 2.4944679374657728), (3, 0.6923553719008263),
 (4, 1.1014462809917351), (5, 0.2696280991735535)]
```

Możesz ponownie wyświetlić zawartość bazy, aby zobaczyć, że współczynnik PageRank dla stron został zaktualizowany:

```
(16, 1.0, 2.49..., 2, 'https://www.dr-chuck.com/csev-blog')
(15, 1.0, 2.13..., 30, 'https://www.dr-chuck.com/csev-blog/2020/08/...')
(15, 1.0, 2.44..., 22, 'https://www.dr-chuck.com/csev-blog/category/...d')
...
22 wierszy.
```

Możesz uruchamiać `sprank.py` tyle razy, ile chcesz, a program po prostu za każdym razem poprawi obliczenie współczynnika PageRank. Możesz uruchomić `sprank.py` nawet kilka razy, następnie dodać kilka kolejnych stron poprzez `spider.py`, a potem znów uruchomić `sprank.py`, żeby dalej przeliczać wartości PageRank. W wyszukiwarce internetowej programy do indeksowania stron, jak i do tworzenia rankingu działają zwykle przez cały czas.

Jeżeli chcesz uruchomić obliczenia PageRank od początku bez ponownego przejścia robotem po stronach, możesz użyć programu `spreset.py`, a następnie ponownie uruchomić `sprank.py`.

Wynik uruchomienia `spreset.py`:

```
Wszystkie strony mają ustawiony współczynnik PageRank na 1.0
```

Ponowne uruchomienie `sprank.py`:

```
Ile iteracji: 50
1 0.720326643053916
2 0.34992366601870745
3 0.17895552923503424
4 0.11665048143652895
...
46 3.579258334100184e-05
47 3.0035450290624035e-05
48 2.520367345856324e-05
49 2.114963873141301e-05
50 1.7747381915049988e-05
[(1, 9.945248881666563e-05), (2, 3.205252622657907), (3, 0.000129079311099),
 (4, 0.0003719906004627465), (5, 9.966636303771006e-05)]
```

Dla każdej iteracji algorytmu PageRank program wypisuje średnią zmianę współczynnika na stronę. Początkowo sieć jest raczej niezbalansowana, więc poszczególne wartości rankingu bardzo się zmieniają pomiędzy kolejnymi iteracjami. Jednak w kilku kolejnych iteracjach algorytm zmierza szybko do końcowego wyniku. Powinieneś uruchomić `sprank.py` na tak długo, by kolejne wartości generowane przez algorytm nie różniły się zbytnio.

Jeśli chcesz zwizualizować strony, które aktualnie znajdują się najwyżej w rankingu, uruchom program `spjson.py`. Odczytuje on bazę danych i zapisuje dane dotyczące najczęściej linkowanych stron w formacie JSON, który da się przejrzeć w przeglądarce internetowej.

```
Tworzenie JSONa w pliku spider.js...
Ile węzłów? 30
Otwórz force.html w przeglądarce internetowej, by zobaczyć wizualizację
```

Możesz obejrzeć wynik, otwierając plik `force.html` w swojej przeglądarce internetowej. Pokazuje on automatyczny układ węzłów (stron) i połączeń między nimi. Możesz kliknąć i przeciągnąć dowolny węzeł, a także dwukrotnie kliknąć na węzeł, by wyświetlić adres URL, który jest przez niego reprezentowany. Wielkość węzła reprezentuje jego istotność, tzn. pokazuje, jak wiele innych stron linkuje do tej strony.

Jeżeli uruchomisz ponownie inne narzędzia tej aplikacji, uruchom też ponownie `spjson.py` i odśwież stronę `force.html` w przeglądarce, aby uzyskać nowe dane umieszczone w `spider.json`.

16.3. Wizualizacja danych z e-maili

W niektórych rozdziałach książki i ćwiczeniach pojawiały się pliki `mbox-short.txt` i `mbox.txt`, które zawierały wiadomości mailowe. Nadszedł czas, by naszą analizę danych dotyczącą poczty elektronicznej przenieść na kolejny poziom.

Zdarza się, że trzeba pobrać z serwerów dane dotyczące poczty e-mail. Może to zająć sporo czasu, a dane mogą być niespójne, pełne błędów i wymagać wielu poprawek lub czyszczenia. W tej sekcji będziemy pracować z najbardziej złożoną jak dotąd aplikacją, pobierzemy prawie gigabajt danych i zwizualizujemy te dane.

Możesz pobrać aplikację z:

Rysunek 16.3. Chmura wyrazów z listy mailingowej deweloperów projektu Sakai

https://py4e.pl/code3/gmane.zip

Będziemy korzystać z danych umieszczonych w bezpłatnej usłudze archiwizacji listy mailingowej o nazwie Gmane[3]. Była ona bardzo popularna wśród twórców projektów open source, ponieważ zapewniała ładne i łatwe do przeszukiwania archiwum ich aktywności mailowej. Posiadała również bardzo liberalną politykę dostępu do swoich danych poprzez swoje API.

Aby nie obciążać innych serwerów, moja własna kopia wiadomości dostępna jest pod adresem:

https://mbox.dr-chuck.net/

Pobieranie dane poczty elektronicznej Sakai za pomocą tej aplikacji wytworzyło prawie gigabajt danych, a cały proces pobierania trwał kilka dni. Plik `README.txt` w powyższym pliku ZIP zawiera instrukcje pobrania archiwalnej kopii `content.sqlite` z większością korpusu tekstowego poczty elektronicznej projektu Sakai (do marca 2015 r.). W związku z tym robot nie musi ściągać danych przez pięć dni. Jeśli pobierzesz ww. archiwalną wersję bazy SQLite, to możesz dodatkowo uruchomić robota internetowego, by uzupełnić dane o najnowsze wiadomości mailowe.

Pierwszym krokiem jest przeskanowanie repozytorium Gmane. Główny adres URL jest zapisany w `gmane.py` w zmiennej `baseurl` i wskazuje na listę mailingową deweloperów projektu Sakai. Twój robot może skanować inne repozytorium – wystarczy, że zmienisz wspomniany adres adres URL. Jeśli to zrobisz, skasuj również plik `content.sqlite`.

Plik `gmane.py` działa odpowiedzialnie – zapisuje w pamięci pobrane dane i odczekuje sekundę po pobraniu stu wiadomości. Program przechowuje wszystkie swoje dane w bazie, może być przerywany i uruchamiany ponownie tak często, jak to konieczne. Pobranie wszystkich danych może potrwać wiele godzin. Może być więc konieczne kilkukrotne ponowne uruchomienie tego programu.

Oto wynik uruchomienia `gmane.py`, który pobiera dziesięć ostatnich wiadomości z listy deweloperów projektu Sakai:

```
Ile wiadomości: 10
http://mbox.dr-chuck.net/sakai.devel/59643/59644 17553
    matthew@longsight.com 2015-03-20T16:27:12-04:00
    re: [building sakai] sakai 10 bulding error
http://mbox.dr-chuck.net/sakai.devel/59644/59645 13128
```

[3] https://en.wikipedia.org/wiki/Gmane

```
    alberto.olivamolina@gmail.com 2015-03-20T16:36:12+01:00
    re: [building sakai] sakai 10 bulding error
http://mbox.dr-chuck.net/sakai.devel/59645/59646 7557
    eric.duquenoy@univ-littoral.fr 2015-03-20T16:52:24+01:00
    [building sakai] lti and sakai groups (or sections)
http://mbox.dr-chuck.net/sakai.devel/59646/59647 1
...
```

Program skanuje zawartość `content.sqlite` w poszukiwaniu numeru pierwszej niepobranej jeszcze wiadomości. Robot ściąga dane tak długo, aż pobierze pożądaną liczbę wiadomości lub gdy dotrze do strony, która nie wydaje się być odpowiednio sformatowaną wiadomością.

Czasami może brakować pewnych wiadomości. Być może administratorzy Gmane usuwali niektóre, a może same gdzieś przepadły. Jeżeli Twój robot się zatrzyma, a wydaje się, że trafił na właśnie taką brakującą wiadomość, to przejdź do przeglądarki bazy SQLite i dodaj wiersz z brakującym id, pozostawiając wszystkie pozostałe pola puste, a następnie uruchom ponownie `gmane.py`. Dzięki temu robot będzie mógł kontynuować pracę. Wstawione ręcznie puste wiadomości będą ignorowane w następnej fazie procesu.

Na szczęście gdy już pobierzesz wszystkie wiadomości i będziesz je miał w `content.sqlite`, to po jakimś czasie będziesz mógł ponownie uruchomić `gmane.py`, i uzyskać tylko nowe wiadomości, które zostały ostatnio wysłane na listę mailingową.

Dane zawarte w bazie `content.sqlite` są dość surowe, oparte na niewydajnym modelu danych, a ponadto nie są skompresowane. Jest to celowe, ponieważ pozwala Ci obejrzeć plik `content.sqlite` w przeglądarce bazy SQLite, by usunąć problemy związane z procesem pobierania danych. Generalnie uruchomianie jakichkolwiek zapytań SQL na tej bazie danych to zły pomysł, ponieważ ich wykonanie byłyby dość powolne.

Druga faza polega na uruchomieniu programu `gmodel.py`. Program ten odczytuje surowe dane z `content` ↪ `.sqlite` i tworzy w pliku `index.sqlite` oczyszczoną i dobrze zamodelowaną wersję danych. Plik ten będzie znacznie mniejszy (często ok. 10 razy mniejszy) niż `content.sqlite`, ponieważ kompresuje on również nagłówek i treść wiadomości.

Przy każdym uruchomieniu `gmodel.py` program usuwa i przebudowuje plik `index.sqlite`, pozwalając Ci dostosować jego parametry i edytować tabele mapowania w `content.sqlite`, tak aby dopasować proces czyszczenia danych. Poniżej mamy przykładowe uruchomienie `gmodel.py`. Wypisuje on linię za każdym razem po przetworzeniu 250 wiadomości mailowych, dzięki czemu widzisz postęp pracy. Jest to istotne, ponieważ przetwarzanie prawie gigabajta danych wejściowych może zająć dłuższy czas.

```
Załadowano nadawców: 1588 , mapowania: 29 , mapowania dns: 1
1 2005-12-08T23:34:30-06:00 ggolden22@mac.com
251 2005-12-22T10:03:20-08:00 tpamsler@ucdavis.edu
501 2006-01-12T11:17:34-05:00 lance@indiana.edu
751 2006-01-24T11:13:28-08:00 vrajgopalan@ucmerced.edu
...
```

Program `gmodel.py` realizuje szereg zadań związanych z czyszczeniem danych.

Nazwy domen *.com*, *.org*, *.edu* i *.net* są przycinane do dwóch poziomów. Inne nazwy domen, do trzech. Tak więc *si.umich.edu* staje się *umich.edu*, a *caret.cam.ac.uk* staje się *cam.ac.uk*. W adresach e-mail wymuszany jest zapis małymi literami, a niektóre z adresów *@gmane.org*, jak np.

```
arwhyte-63aXycvo3TyHXe+LvDLADg@public.gmane.org
```

są konwertowane za każdym razem, gdy rzeczywisty e-mail znajduje się w korpusie wiadomości.

W bazie danych `mapping.sqlite` znajdują się dwie tabele pozwalające na mapowanie zarówno nazw domen, jak i poszczególnych adresów e-mail, które zmieniają się w ciągu całego okresu życia listy mailingowej. Na przykład Steve Githens używał następujących adresów e-mail, ponieważ zmieniał pracę:

```
s-githens@northwestern.edu
sgithens@cam.ac.uk
swgithen@mtu.edu
```

Jeśli chcemy, to możemy dodać dwa wpisy do tabeli mapowania nadawców *Mapping* w `mapping.sqlite`, tak by `gmodel.py` mapował wszystkie trzy adresy na jeden:

```
s-githens@northwestern.edu -> swgithen@mtu.edu
sgithens@cam.ac.uk -> swgithen@mtu.edu
```

Jeżeli istnieje wiele nazw DNS, które chcesz zmapować na jeden DNS, to możesz również dokonać podobnych wpisów w tabeli *DNSMapping*. Przykładowo:

```
iupui.edu -> indiana.edu
```

dzięki czemu wszystkie konta z różnych kampusów Uniwersytetu Indiany będą śledzone razem.

Możesz uruchamiać `gmodel.py` wielokrotnie, przeglądać dane i dodawać mapowania, dzięki czemu dane do analizy będą coraz bardziej przejrzyste. Kiedy skończysz, w pliku `index.sqlite` będziesz miał ładnie zaindeksowaną wersję e-maili. Jest to plik, którego możesz użyć do dalszej analizy danych. Z tym plikiem analiza będzie naprawdę szybka.

Pierwszą i najprostszą analizą jest określenie "kto wysłał najwięcej wiadomości?" i "która organizacja wysłała najwięcej listów?". Odpowiedzi na te pytania uzyskamy za pomocą programu `gbasic.py`:

```
Ile wyświetlić? 5
Załadowanych wiadomości= 51330 tematów= 25033 nadawców= 1584

Lista 5 najczęstszych użytkowników
steve.swinsburg@gmail.com 2657
azeckoski@unicon.net 1742
ieb@tfd.co.uk 1591
csev@umich.edu 1304
david.horwitz@uct.ac.za 1184

Lista 5 najczęstszych organizacji
gmail.com 7339
umich.edu 6243
uct.ac.za 2451
indiana.edu 2258
unicon.net 2055
```

Zauważ, jak szybko działa `gbasic.py` w porównaniu z `gmane.py`, a nawet z `gmodel.py`. Wszystkie pracują na tych samych danych, ale `gbasic.py` używa skompresowanych i znormalizowanych danych znajdujących się w `index.sqlite`. Jeśli masz dużo danych do przetworzenia, to wieloetapowy proces tworzenia narzędzi do analizy (taki jak w tej aplikacji) może zająć Ci trochę więcej czasu, ale z drugiej strony zaoszczędzi Ci go bardzo dużo, gdy naprawdę zaczniesz eksplorować i wizualizować swoje dane.

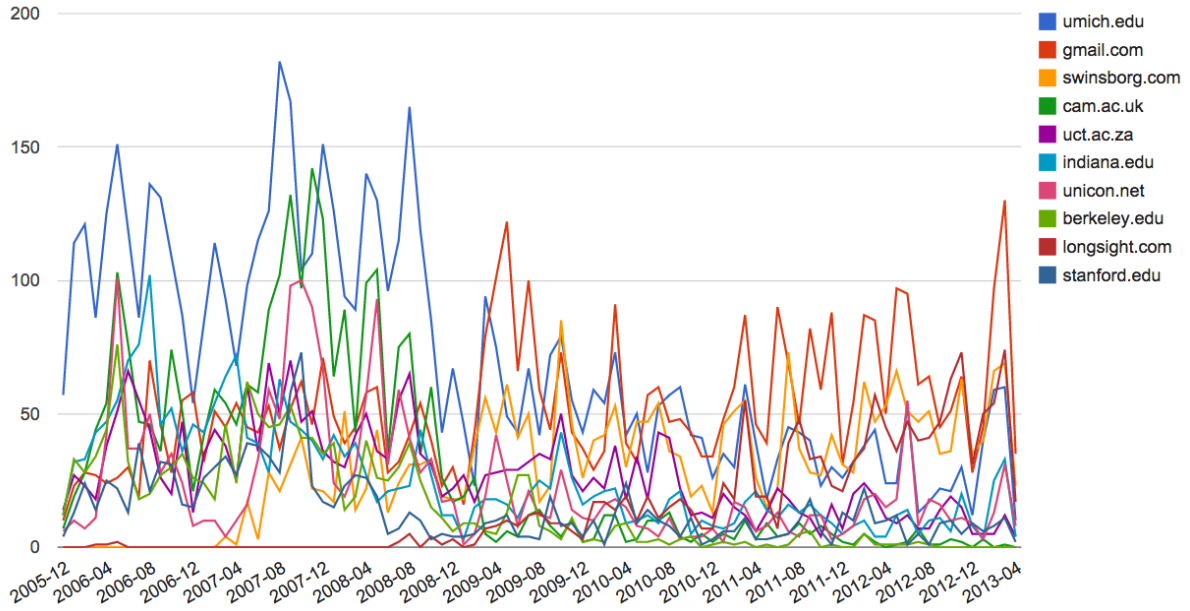
Poprzez program `gword.py` możesz stworzyć prostą wizualizację częstości występowania słów w tematach wiadomości:

```
Zakres częstości: 33229 129
Wynik zapisano w gword.js
Otwórz gword.htm w przeglądarce internetowej by zobaczyć wizualizację
```

W ten sposób powstaje plik `gword.js`, który możesz zwizualizować za pomocą `gword.htm`. Mamy utworzoną chmurę wyrazów podobną do tej, która znajduje się w grafice na początku tej sekcji.

Druga wizualizacja jest tworzona przez program `gline.py`. Oblicza ona udział organizacji w wiadomościach mailowych w danym czasie.

```
Loaded messages= 51330 subjects= 25033 senders= 1584
Top 10 organizacji
['gmail.com', 'umich.edu', 'uct.ac.za', 'indiana.edu',
'unicon.net', 'tfd.co.uk', 'berkeley.edu', 'longsight.com',
'stanford.edu', 'ox.ac.uk']
```



Rysunek 16.4. Aktywność mailowa w projekcie Sakai z podziałem na organizacje

```
Wynik zapisano w gline.js
Otwórz gline.htm by zwizualizować dane
```

Wynik jest zapisywany w pliku `gline.js`, który można zwizualizować przy użyciu strony `gline.htm`.

Podsumowując, jest to stosunkowo złożona i wyrafinowana aplikacja, posiadająca funkcje pozwalające na wyszukiwanie, czyszczenie i wizualizację prawdziwych danych.